

OpenEye
SCIENTIFIC

OEChem TK – C#

Release 2026.1.0

Cadence Design Systems, Inc.

August 06, 2026

CONTENTS

1 Introduction **1**
- 1.1 How to read this manual . . . 1
- 1.2 OEChem and Informatics . . . 1

2 Theory **3**
- 2.1 OEChem Theory . . . 3
  - 2.1.1 Molecules . . . 3
  - 2.1.2 Molecule Reading and Writing . . . 13
  - 2.1.3 Tagged Data Manipulation . . . 24
  - 2.1.4 Atom and Bond Traversal . . . 35
  - 2.1.5 Atom, Bond, and Conformer Indices . . . 40
  - 2.1.6 Atom Properties . . . 43
  - 2.1.7 Bond Properties . . . 46
  - 2.1.8 Valence Models . . . 47
  - 2.1.9 Molecule Properties . . . 50
  - 2.1.10 Ring Perception . . . 54
  - 2.1.11 Aromaticity Perception . . . 60
  - 2.1.12 Connectivity Perception . . . 64
  - 2.1.13 Stereochemistry Perception . . . 68
  - 2.1.14 Formal and Partial Charges . . . 79
  - 2.1.15 2D Coordinate Generation . . . 81
  - 2.1.16 Molecule Styling . . . 84
  - 2.1.17 Pattern Matching . . . 85
  - 2.1.18 Substructure Search with MDL Queries . . . 103
  - 2.1.19 Reactions . . . 117
  - 2.1.20 Multi-conformer Molecules . . . 136
  - 2.1.21 Molecular Database Handling . . . 145
  - 2.1.22 Design Decisions . . . 149
  - 2.1.23 Multi-Threading . . . 155
  - 2.1.24 Predicate Functors . . . 164
  - 2.1.25 OEMolRecord . . . 176
  - 2.1.26 Periodic Table Functions . . . 176
  - 2.1.27 SMILES Line Notation . . . 178
  - 2.1.28 SMARTS Pattern Matching . . . 180
  - 2.1.29 Atom Stereo From Bond Stereo . . . 183
  - 2.1.30 InChI validation . . . 189
- 2.2 OESystem Theory . . . 190
  - 2.2.1 Command Line Interface and Options . . . 190
  - 2.2.2 Command Line Interface Definition . . . 196
  - 2.2.3 Generic Data . . . 206

2.2.4 OERecord . . . 210
2.3 OEBio Theory . . . 211
2.3.1 Biopolymers . . . 211
2.3.2 Protein Preparation . . . 218
2.3.3 Macromolecule Conformations . . . 225
2.3.4 Interaction Perception . . . 227
2.3.5 Receptors . . . 231
2.4 OEGrid Theory . . . 233
2.4.1 Scalar Grids . . . 233

3 Examples **241**
3.1 OEChem Examples . . . 241
3.1.1 Molecule Processing . . . 241
3.1.2 Reactions . . . 241
3.1.3 Molecule Searching . . . 241
3.1.4 Molecule Alignment . . . 242
3.1.5 SDF Specific . . . 242
3.1.6 Molecule Information Examples . . . 242
3.1.7 2D Coordinate Generation Examples . . . 242
3.1.8 3D Molecule Styling Examples . . . 242
3.1.9 Miscellaneous Examples . . . 242
3.2 OEBio Examples . . . 349
3.2.1 Extracting Protein Backbone . . . 349
3.2.2 Extracting Ligand from *OEDesignUnit* . . . 351
3.2.3 Printing protein amino acids histograms . . . 351
3.2.4 Splitting a Macro-molecular Complex . . . 353
3.2.5 Splitting a Macro-molecular Complex Efficiently and Flexibly . . . 357
3.2.6 Split a Macro-molecular Complex Into Fragments . . . 361
3.2.7 Preparing a Protein . . . 364
3.2.8 Perceive and Print Protein-Ligand Interactions . . . 373
3.3 OEMath Examples . . . 378
3.3.1 Fitting model with kernel PLS . . . 378
3.4 Python Cookbook Examples . . . 378

4 API **381**
4.1 OEChem API . . . 381
4.1.1 OEChem Classes . . . 381
4.1.2 OEChem Constants . . . 658
4.1.3 OEChem Functions . . . 878
4.2 Preliminary OEChem API . . . 1018
4.2.1 Preliminary OEChem Classes . . . 1018
4.2.2 Preliminary OEChem Functions . . . 1041
4.3 OESystem API . . . 1057
4.3.1 OESystem Classes . . . 1057
4.3.2 OESystem Constants . . . 1187
4.3.3 OESystem Functions . . . 1200
4.4 Preliminary OESystem API . . . 1244
4.4.1 Preliminary OESystem Classes . . . 1244
4.4.2 Preliminary OESystem Constants . . . 1261
4.4.3 Preliminary OESystem Functions . . . 1262
4.5 OEPlatform API . . . 1264
4.5.1 OEPlatform Classes . . . 1264
4.5.2 OEPlatform Constants . . . 1291
4.5.3 OEPlatform Functions . . . 1292

4.6 OEMath API . . . 1301
4.6.1 OEMath Classes . . . 1301
4.6.2 OEMath Constants . . . 1308
4.6.3 OEMath Functions . . . 1310
4.7 OEBio API . . . 1324
4.7.1 OEBio Classes . . . 1324
4.7.2 OEBio Constants . . . 1474
4.7.3 OEBio Functions . . . 1508
4.8 Preliminary OEBio API . . . 1543
4.8.1 Preliminary OEBio Functions . . . 1543

5 Release History **1547**
5.1 OEChem TK 5.0.0 . . . 1547
5.1.1 New Features . . . 1547
5.1.2 Minor Bug Fixes . . . 1549
5.1.3 Python-Specific Changes . . . 1549
5.2 OEBio TK . . . 1549
5.2.1 New Features . . . 1549
5.2.2 Major Bug Fixes . . . 1551
5.2.3 Minor Bug Fixes . . . 1552
5.2.4 Python-Specific Changes . . . 1552
5.3 OEChem TK 4.3.0 . . . 1552
5.3.1 New Features . . . 1552
5.3.2 Major Bug Fixes . . . 1553
5.3.3 Minor Bug Fixes . . . 1553
5.4 OEBio TK 4.3.0 . . . 1553
5.4.1 New Features . . . 1553
5.4.2 Major Bug Fixes . . . 1553
5.4.3 Minor Bug Fixes . . . 1553
5.5 OEGrid 4.3.0 . . . 1553
5.5.1 New Features . . . 1553
5.6 OEChem TK 4.2.0 . . . 1554
5.6.1 New Features . . . 1554
5.6.2 Major Bug Fixes . . . 1554
5.6.3 Minor Bug Fixes . . . 1554
5.6.4 Python-Specific Changes . . . 1554
5.7 OEBio TK 4.2.0 . . . 1554
5.7.1 Minor Bug Fixes . . . 1554
5.8 OEChem TK 4.1.1.1 . . . 1554
5.8.1 Major bug fixes . . . 1554
5.9 OEChem TK 4.1.1.0 . . . 1555
5.9.1 New features . . . 1555
5.9.2 Minor bug fixes . . . 1555
5.10 OEPlatform TK 4.1.1 . . . 1555
5.10.1 New features . . . 1555
5.11 OEBio TK 4.1.1 . . . 1555
5.11.1 Major bug fixes . . . 1555
5.12 OEChem TK 4.1.0 . . . 1556
5.12.1 New features . . . 1556
5.12.2 Major bug fixes . . . 1557
5.12.3 Minor bug fixes . . . 1557
5.13 OESystem TK 4.1.0 . . . 1557
5.13.1 New features . . . 1557
5.14 OEMath TK 4.1.0 . . . 1558

5.14.1 New features . . . 1558
5.15 OEBio TK 4.1.0 . . . 1558
5.15.1 New features . . . 1558
5.15.2 Major bug fixes . . . 1558
5.15.3 Minor bug fixes . . . 1558
5.16 OEGrid TK 4.1.0 . . . 1558
5.16.1 Major bug fixes . . . 1558
5.17 OESystem TK 4.1.0 . . . 1558
5.17.1 New features . . . 1558
5.18 OEMath TK 4.1.0 . . . 1559
5.18.1 New features . . . 1559
5.19 OEChem TK 4.0.0 . . . 1559
5.19.1 New features . . . 1559
5.19.2 Minor bug fixes . . . 1560
5.20 OEBio TK 4.0.0 . . . 1560
5.20.1 New features . . . 1560
5.21 OESystem TK 4.0.0 . . . 1561
5.21.1 New features . . . 1561
5.22 OEPlatform TK 4.0.0 . . . 1561
5.23 OEGrid TK 4.0.0 . . . 1561
5.24 OEChem TK 3.4.0 . . . 1561
5.24.1 New features . . . 1561
5.24.2 Minor bug fixes . . . 1561
5.24.3 Python-specific changes . . . 1562
5.25 OESystem TK 3.3.1 . . . 1562
5.26 OEPlatform TK 3.3.1 . . . 1562
5.26.1 C++-specific changes . . . 1562
5.27 OEChem TK 3.3.1 . . . 1562
5.27.1 New features . . . 1562
5.27.2 Major bug fixes . . . 1562
5.27.3 Minor bug fixes . . . 1562
5.28 OEBio TK 3.3.1 . . . 1562
5.28.1 New features . . . 1562
5.28.2 Minor bug fixes . . . 1562
5.29 OESystem TK 3.3.1 . . . 1563
5.29.1 New features . . . 1563
5.29.2 Major bug fixes . . . 1563
5.29.3 Minor bug fixes . . . 1563
5.30 OEPlatform TK 3.3.1 . . . 1563
5.31 OEChem TK 3.3.0 . . . 1563
5.31.1 New features . . . 1563
5.31.2 Minor bug fixes . . . 1563
5.32 OEBio TK 3.3.0 . . . 1564
5.32.1 Minor bug fixes . . . 1564
5.33 OESystem TK 3.3.0 . . . 1564
5.33.1 New features . . . 1564
5.33.2 Minor bug fixes . . . 1564
5.34 OEPlatform TK 3.3.0 . . . 1564
5.34.1 General notices . . . 1564
5.35 OEGrid TK 3.3.0 . . . 1564
5.35.1 New features . . . 1564
5.36 OEChem TK 3.2.0 . . . 1564
5.36.1 New features . . . 1564
5.36.2 Major bug fixes . . . 1565

5.36.3 Minor bug fixes . . . 1565
5.37 OEBio TK 3.2.0 . . . 1565
5.37.1 New features . . . 1565
5.37.2 Minor bug fixes . . . 1565
5.38 OESystem TK 3.2.0 . . . 1566
5.38.1 New features . . . 1566
5.38.2 Minor bug fixes . . . 1566
5.39 OEPlatform TK 3.2.0 . . . 1566
5.40 OEGrid TK 3.2.0 . . . 1566
5.41 OEMath TK 3.2.0 . . . 1566
5.41.1 New features . . . 1566
5.42 OEChem TK 3.1.1 . . . 1566
5.42.1 New features . . . 1566
5.42.2 Major bug fixes . . . 1566
5.42.3 Minor bug fixes . . . 1567
5.42.4 Documentation changes . . . 1567
5.43 OEBio TK 3.1.1 . . . 1567
5.43.1 New features . . . 1567
5.43.2 Minor bug fixes . . . 1567
5.44 OESystem TK 3.1.1 . . . 1568
5.44.1 Major bug fixes . . . 1568
5.45 OEPlatform TK 3.1.1 . . . 1568
5.45.1 New features . . . 1568
5.46 OEGrid TK 3.1.1 . . . 1568
5.46.1 Major bug fixes . . . 1568
5.47 OEMath TK 3.1.1 . . . 1568
5.48 Version 3.1.0.2 . . . 1568
5.48.1 OEChem TK 3.1.0.2 . . . 1568
5.48.2 OEBio TK 3.1.0.2 . . . 1568
5.49 Version 3.1.0 . . . 1569
5.49.1 OEChem TK 3.1.0 . . . 1569
5.49.2 OEBio TK 3.1.0 . . . 1569
5.49.3 OESystem TK 3.1.0 . . . 1570
5.49.4 OEPlatform TK 3.1.0 . . . 1570
5.49.5 OEGrid TK 3.1.0 . . . 1571
5.49.6 OEMath TK 3.1.0 . . . 1571
5.50 Version 3.0.0.6 . . . 1572
5.50.1 OEChem TK 3.0.0.6 . . . 1572
5.50.2 OESystem TK 3.0.0.6 . . . 1572
5.51 Version 3.0.0.5 . . . 1572
5.51.1 OEChem TK 3.0.0.5 . . . 1572
5.51.2 OESystem TK 3.0.0.5 . . . 1572
5.51.3 OEGrid TK 3.0.0.5 . . . 1572
5.52 Version 3.0.0 . . . 1572
5.52.1 OEChem TK 3.0.0 . . . 1572
5.52.2 OEBio TK 3.0.0 . . . 1573
5.52.3 OESystem TK 3.0.0 . . . 1573
5.52.4 OEPlatform TK 3.0.0 . . . 1574
5.52.5 OEGrid TK 3.0.0 . . . 1574
5.53 Version 2.3.0 . . . 1574
5.53.1 OEChem TK 2.3.0 . . . 1574
5.53.2 OEBio TK 2.3.0 . . . 1577
5.53.3 OESystem TK 2.3.0 . . . 1578
5.53.4 OEPlatform TK 2.3.0 . . . 1578

5.53.5 OEGrid TK 1.6.4 . . . 1578
5.54 Version 2.2.0 . . . 1578
5.54.1 OEChem TK 2.2.0 . . . 1578
5.54.2 OEBio TK 2.2.0 . . . 1581
5.54.3 OESystem TK 2.2.0 . . . 1582
5.54.4 OEPlatform TK 2.2.0 . . . 1582
5.54.5 OEGrid TK 1.6.3 . . . 1583
5.55 Version 2.1.5 . . . 1583
5.55.1 OEChem TK 2.1.5 . . . 1583
5.55.2 OEBio TK 2.1.5 . . . 1584
5.55.3 OEPlatform TK 2.1.5 . . . 1585
5.55.4 OESystem TK 2.1.5 . . . 1585
5.56 Version 2.1.4 . . . 1585
5.56.1 OEChem TK 2.1.4 . . . 1585
5.56.2 OESystem TK 2.1.4 . . . 1587
5.56.3 OEBio TK 2.1.4 . . . 1587
5.57 Version 2.1.3 . . . 1587
5.57.1 OEChem TK 2.1.3 . . . 1587
5.57.2 OESystem TK 2.1.3 . . . 1588
5.57.3 OEBio TK 2.1.3 . . . 1589
5.57.4 OEGrid TK 1.6.0 . . . 1589
5.58 Version 2.1.2 . . . 1590
5.58.1 OEChem TK 2.1.2 . . . 1590
5.58.2 OEBio TK 2.1.2 . . . 1591
5.59 Version 2.1.1 . . . 1592
5.59.1 OEChem 2.1.1 . . . 1592
5.59.2 OEBio 2.1.1 . . . 1593
5.59.3 OEPlatform 2.1.1 . . . 1594
5.59.4 OESystem 2.1.1 . . . 1594
5.60 Version 2.1.0 . . . 1594
5.60.1 OEChem 2.1.0 . . . 1594
5.60.2 OEBio 2.1.0 . . . 1595
5.60.3 OEPlatform 2.1.0 . . . 1596
5.61 Version 2.0.7 . . . 1596
5.61.1 OEChem 2.0.7 . . . 1596
5.61.2 OEBio 2.0.7 . . . 1598
5.61.3 OESystem 2.0.7 . . . 1600
5.61.4 OEPlatform 2.0.7 . . . 1600
5.62 Version 2.0.6 . . . 1600
5.62.1 OEChem 2.0.6 . . . 1600
5.62.2 OEBio 2.0.6 . . . 1603
5.62.3 OESystem 2.0.6 . . . 1604
5.62.4 OEPlatform 2.0.6 . . . 1604
5.62.5 OEGrid 1.5.3 . . . 1604
5.63 Version 2.0.5 . . . 1604
5.63.1 OEChem 2.0.5 . . . 1604
5.63.2 OEBio 2.0.5 . . . 1609
5.63.3 OESystem 2.0.5 . . . 1610
5.63.4 OEPlatform 2.0.5 . . . 1610
5.63.5 OEGrid . . . 1611
5.64 Version 2.0.4 . . . 1611
5.64.1 OEChem 2.0.4 . . . 1611
5.64.2 OEBio 2.0.4 . . . 1614
5.64.3 OESystem 2.0.4 . . . 1615

5.64.4 OEPlatform 2.0.4 . . . 1615
5.64.5 OEGrid 1.5.1 . . . 1616
5.65 Version 2.0.3 . . . 1616
5.65.1 OEChem 2.0.3 . . . 1616
5.65.2 OEBio 2.0.3 . . . 1618
5.65.3 OESystem 2.0.3 . . . 1619
5.65.4 OEPlatform 2.0.3 . . . 1619
5.65.5 OEGrid 1.5.0 . . . 1619
5.66 Version 2.0.2 . . . 1619
5.66.1 OEChem 2.0.2 . . . 1619
5.66.2 OEBio 2.0.2 . . . 1624
5.66.3 OESystem 2.0.2 . . . 1624
5.66.4 OEPlatform 2.0.2 . . . 1625
5.66.5 OEGrid 1.4.7 . . . 1625
5.67 Version 2.0.1 . . . 1625
5.67.1 OEChem 2.0.1 . . . 1625
5.67.2 OEBio 2.0.1 . . . 1627
5.67.3 OESystem 2.0.1 . . . 1628
5.67.4 OEPlatform 2.0.1 . . . 1628
5.67.5 OEGrid 1.4.6 . . . 1629
5.68 Version 2.0.0 . . . 1629
5.68.1 OEChem 2.0.0 . . . 1629
5.68.2 OESystem 2.0.0 . . . 1632
5.68.3 OEPlatform 2.0.0 . . . 1633
5.68.4 OEGrid 1.4.5 . . . 1633
5.69 Version 1.9.3 . . . 1634
5.69.1 OEChem 1.9.3 . . . 1634
5.69.2 OESystem 1.9.3 . . . 1635
5.69.3 OEPlatform 1.9.3 . . . 1635
5.69.4 OEGrid 1.4.4 . . . 1635
5.70 Version 1.9.2 . . . 1635
5.70.1 OEChem 1.9.2 . . . 1635
5.70.2 OESystem 1.9.2 . . . 1638
5.70.3 OEPlatform 1.9.2 . . . 1638
5.70.4 OEBio 1.9.2 . . . 1638
5.70.5 OEGrid 1.4.3 . . . 1638
5.71 Version 1.9.1 . . . 1639
5.71.1 OEChem 1.9.1 . . . 1639
5.71.2 OESystem 1.9.1 . . . 1640
5.71.3 OEPlatform 1.9.1 . . . 1640
5.71.4 OEBio 1.9.1 . . . 1641
5.71.5 OEGrid 1.4.2 . . . 1641
5.72 Version 1.9.0 . . . 1641
5.72.1 OEChem 1.9.0 . . . 1641
5.72.2 OESystem 1.9.0 . . . 1642
5.72.3 OEPlatform 1.9.0 . . . 1643
5.72.4 OEBio 1.9.0 . . . 1643
5.72.5 OEGrid 1.4.1 . . . 1643
5.73 Version 1.8.0 . . . 1643
5.73.1 OEChem 1.8.0 . . . 1643
5.73.2 OEBio 1.8.0 . . . 1644
5.73.3 OEGrid 1.4.0 . . . 1644
5.73.4 OESystem 1.8.0 . . . 1645
5.73.5 OEPlatform 1.8.0 . . . 1645

5.74 Version 1.7.7 . . . 1646
5.74.1 OEChem 1.7.7 . . . 1646
5.74.2 OEBio 1.7.7 . . . 1646
5.74.3 OESystem 1.7.7 . . . 1647
5.74.4 OEPlatform 1.7.7 . . . 1647
5.75 Version 1.7.6 . . . 1647
5.75.1 OESystem 1.7.6 . . . 1647
5.75.2 OEBio 1.7.6 . . . 1648
5.76 Version 1.7.5 . . . 1648
5.76.1 OEChem 1.7.5 . . . 1648
5.76.2 OEBio 1.7.5 . . . 1649
5.76.3 OEGrid 1.3.5 . . . 1650
5.76.4 OESystem 1.7.5 . . . 1650
5.76.5 OEPlatform 1.7.5 . . . 1651
5.77 Version 1.7.4 . . . 1651
5.77.1 OEChem 1.7.4 . . . 1651
5.77.2 OESystem 1.7.4 . . . 1652
5.77.3 OEPlatform 1.7.4 . . . 1652
5.77.4 OEBio 1.7.4 . . . 1652
5.77.5 OEGrid 1.3.4 . . . 1652
5.78 Version 1.7.3 . . . 1652
5.78.1 OEChem 1.7.3 . . . 1652
5.79 Version 1.7.2 . . . 1653
5.79.1 OEChem 1.7.2 . . . 1653
5.79.2 OESystem 1.7.2 . . . 1654
5.79.3 OEPlatform 1.7.2 . . . 1654
5.79.4 OEBio 1.7.2 . . . 1655
5.79.5 OEGrid 1.3.3 . . . 1655
5.80 Version 1.7.1 . . . 1655
5.80.1 OEChem 1.7.1 . . . 1655
5.81 Version 1.7.0 . . . 1656
5.81.1 OEChem 1.7.0 . . . 1656
5.81.2 OEGrid 1.3.2 . . . 1659
5.81.3 OESystem 1.7.0 . . . 1659
5.81.4 OEPlatform 1.7.0 . . . 1660
5.82 Version 1.6.1 . . . 1660
5.82.1 OEChem 1.6.1 . . . 1660
5.82.2 OESystem 1.6.1 . . . 1661
5.82.3 OEPlatform 1.6.1 . . . 1661
5.83 Version 1.6.0 . . . 1661
5.83.1 OEChem 1.6.0 . . . 1661
5.83.2 OESystem 1.6.0 . . . 1665
5.83.3 OEPlatform 1.6.0 . . . 1665
5.84 Version 1.5.1 . . . 1665
5.84.1 OEChem 1.5.1 . . . 1665
5.85 Version 1.5.0 . . . 1666
5.85.1 OEChem 1.5.0 . . . 1666
5.86 Version 1.4.2 . . . 1668
5.86.1 OEChem 1.4.2 . . . 1668
5.86.2 OEBio 1.4.2 . . . 1670
5.86.3 OESystem 1.4.2 . . . 1670
5.86.4 OEPlatform 1.4.2 . . . 1671
5.87 Version 1.4.1 . . . 1671
5.87.1 OEChem 1.4.1 . . . 1671

5.88 Version 1.4.0 . . . 1671
5.88.1 OEChem 1.4.0 . . . 1671
5.88.2 OEBio 1.4.0 . . . 1674
5.88.3 OESystem 1.4.0 . . . 1674
5.88.4 OEPlatform 1.4.0 . . . 1675
5.89 Version 1.3.4 . . . 1675
5.89.1 OEChem 1.3.4 . . . 1675
5.89.2 OESystem 1.3.4 . . . 1677
5.89.3 OEPlatform 1.3.4 . . . 1678
5.90 Version 1.3.3 . . . 1678
5.90.1 OEChem 1.3.3 . . . 1678
5.90.2 OESystem 1.3.3 . . . 1681
5.90.3 OEPlatform 1.3.3 . . . 1682
5.91 Version 1.3.2 . . . 1682
5.91.1 OEChem 1.3.2 . . . 1682
5.91.2 OESystem 1.3.2 . . . 1684
5.91.3 OEPlatform 1.3.2 . . . 1685
5.92 Version 1.3.1 . . . 1685
5.92.1 OEChem 1.3.1 . . . 1685
5.92.2 OESystem 1.3.1 . . . 1686
5.93 Version 1.3.0 . . . 1686
5.93.1 OEChem 1.3.0 . . . 1686
5.93.2 OESystem 1.3.0 . . . 1689
5.93.3 OEPlatform 1.3.0 . . . 1690

6 Copyright and Trademarks **1691**

7 Sample Code **1693**

8 Citation **1695**
8.1 Orion® . . . 1695
8.2 Orion Floes . . . 1695
8.3 Toolkits and Applications . . . 1695
8.4 Publications for Bibliographies . . . 1697
8.4.1 Orion . . . 1697
8.4.2 AFITT and FLYNN Applications . . . 1697
8.4.3 OEDocking Application and Toolkit . . . 1697
8.4.4 OMEGA Application and Toolkit . . . 1697
8.4.5 ROCS Application . . . 1698
8.5 OpenEye MMDS Web Service . . . 1698

9 Technology Licensing **1699**
9.1 GCC . . . 1771
9.1.1 GCC RUNTIME LIBRARY EXCEPTION . . . 1772
9.1.2 GNU GENERAL PUBLIC LICENSE . . . 1773

Index **1787**

CHAPTER

ONE

INTRODUCTION

This is **OEChem TK**'s C#/.Net Programming manual.

1.1 How to read this manual

This is a collection of prose covering many of the important topics which can be addressed by the **OEChem TK** library. This manual is meant to be read from front to back at least once. Each topic in this manual is introduced assuming the material presented earlier in the manual has been read. Further, the complexity of topics as well as the complexity of the example code grows as the text progresses. While the initial listings are effectively the "Hello World" of **OEChem TK**, later examples may require some time to comprehend fully. This manual is filled with example programs. We encourage you to compile, test, and modify the examples we present.

See also

Experienced **OEChem TK** programmers should use the Application Programming Interface (*OEChem API*) for the most thorough reference of **OEChem TK** functionality. `N.B.` The API section of the documentation is automatically generated from the C++ header files. Due to technical limitations it is not possible to provide each API function with the correct C# syntax. If you have any questions about the correct syntax for a function please contact support (support@eyesopen.com)

1.2 OEChem and Informatics

Chemical information processing is the science of representing molecules in computers. Hence the fundamental "object" or data structure within a chemical information system is that of the molecule, its atoms, and its bonds.

A significant problem encountered in such systems is that different applications place differing requirements or constraints on how a molecule is represented. In protein biochemistry, molecules are divided into amino acid residues, with specific atom naming and conformational information such as alpha helix or beta sheet. Inorganic chemistry requires isotopic and coordination information for atoms and modeling of complex chiralities.

One possible solution is to prescribe a single data structure that encodes all of the potential information required of a molecule. However, such an approach suffers from the fact that "you cannot please all of the chemists, all of the time." A requirement in the field of chemical databases and substructure searching is that a molecule representation be as compact as possible, to allow as much information to be held in memory as possible and maximize the performance of processing databases from disk.

For this reason, the molecule, its atoms, and its bonds are defined in as abstract a manner as possible in **OEChem TK**. The following is a quick guide to the chapters covering these fundamental classes:

OEMolBase

- *Molecules*

- *Molecule Reading and Writing*
- *Molecule Properties*
- *Multi-conformer Molecules*
- *Design Decisions*

OEAtomBase, *OEBondBase*

- *Atom and Bond Traversal*
- *Atom Properties*
- *Bond Properties*
- *Atom, Bond, and Conformer Indices*

See also

- C# Quick Start Manual for installation instructions.
- How to import the toolkits section

CHAPTER

TWO

THEORY

2.1 OEChem Theory

Introduction to Molecules

2.1.1 Molecules

The *OEGraphMol* is the object representing a molecule used in most example programs you will find in **OEChem TK**'s example directories, or in the code examples of this manual. An *OEGraphMol* is a concrete class which can be declared and used for most molecular functions in **OEChem TK**. Much of an *OEGraphMol*'s API is defined by the *OEMolBase* abstract base-class. An *OEGraphMol* can be passed to any function which takes an *OEMolBase* argument.

See also

An `OEGraphMol` contains atoms and bonds. Their access is discussed in chapter *Atom and Bond Traversal*.

Construction and Destruction

The example below represents the smallest possible C# **OEChem TK** program. This program creates an *OEGraphMol* called `mol` when run. When the program ends, C# automatically cleans up the molecule when there are no more references to it.

Create a molecule

```
using System;
using OpenEye.OEChem;

public class CreateOEGraphMol
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
    }
}
```

Construction from SMILES

A common method of creating a molecule in **OEChem TK** is via the SMILES representation. SMILES notation is commonly used in chemical information systems, as it provides a convenient string representation of a molecule. An introduction to SMILES syntax is provided in chapter *SMILES Line Notation*. The following examples will use the SMILES `c1ccccc1` which describes the molecule benzene. A molecule can be created from a SMILES

string using the *OESmilesToMol* function. Similarly, a molecule can be created from a CXSMILES string using the *OECXSMILESToMol*

Creating a molecule from a SMILES string (version 1)

```
using System;
using OpenEye.OEChem;

public class CreateOEGraphMolFromSMILES
{
    public static void Main(string[] argv)
    {
        // create a new molecule
        OEGraphMol mol = new OEGraphMol();

        // convert the SMILES string into a molecule
        OEChem.OESmilesToMol(mol, "c1ccccc1");
    }
}
```

The *OESmilesToMol* function returns a boolean value indicating whether the input string was a valid SMILES representation of a molecule. It is good programming practice to check the return value and report an error message if anything went wrong. The following example shows adding a check on the return status of *OESmilesToMol* and printing an error message if the string was not a valid SMILES representation of a molecule.

Creating a molecule from a SMILES string (version 2)

```
using System;
using OpenEye.OEChem;

public class OEGraphMolFromSMILESCheck
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();

        if (OEChem.OESmilesToMol(mol, "c1ccccc1"))
        {
            // do something with the molecule
        }
        else
        {
            OEChem.OEThrow.Warning("SMILES string was invalid!");
        }
    }
}
```

The *OESmilesToMol* is considered a **high-level** function. In addition, to parsing the given SMILES string, the *OESmilesToMol* function also perceives:

- the rings of the molecule, by invoking the *OEFindRingAtomsAndBonds* function
- the aromaticity of the molecule, by calling the *OEAssignAromaticFlags* function using the *OEAroModel.OpenEye* aromaticity model
- the chirality of the molecule, by calling the *OEPerceiveChiral* function

In cases where you want to preserve the aromaticity of the SMILES string (or the lack of it), a **low-level** *OEParseSmiles* function can be used. For example, if benzene is expressed as `c1ccccc1`, all atoms and bonds are marked as aromatic. But if it is expressed as a Kekulé form, `C1=CC=CC=C1`, all atoms and bonds are kept aliphatic. The aromaticity of the molecule can be perceived by calling the *OEAssignAromaticFlags* function.

Creating molecules from a SMILES string (version 3)

```
using System;
using OpenEye.OEChem;

public class OEGraphMolFromSMILESAromatic
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();

        if (!OEChem.OEParseSmiles(mol, "c1ccccc1"))
        {
            OEChem.OEThrow.Warning("SMILES string was invalid!");
        }

        Console.WriteLine("Number of aromatic atoms: " + OEChem.OECount(mol, new␣
↪OEIsAromaticAtom()));
        OEChem.OEAssignAromaticFlags(mol);
        Console.WriteLine("Number of aromatic atoms: " + OEChem.OECount(mol, new␣
↪OEIsAromaticAtom()));
    }
}
```

The output of the preceding program is the following:

```
Number of aromatic atoms = 0
Number of aromatic atoms = 6
```

Hint

We highly recommend the use of the *OESmilesToMol* function when creating a molecule from a SMILES string.

We highly recommend the use of the *OECXSMILESToMol* function when creating a molecule from either a SMILES or CXSMILES string and the specific format is not known.

See also

- *Aromaticity Perception* chapter for further information about aromaticity models.
- *OESmilesToMol*
- *OECXSMILESToMol*

Reuse

Consider the following code examples to parse two separate SMILES strings, benzene and phenol, and print the number of heavy atoms in each.

Reusing a molecule (OESmilesToMol)

```
using System;
using OpenEye.OEChem;

public class ReuseMoleculeOESmilesToMol
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();

        OEChem.OESmilesToMol(mol, "c1ccccc1");
        Console.WriteLine("Number of benzene atoms: " + mol.NumAtoms());

        OEChem.OESmilesToMol(mol, "c1ccccc1O");
        Console.WriteLine("Number of phenol atoms: " + mol.NumAtoms());
     }
}
```

The **high-level** *OESmilesToMol* function automatically clears the molecule before parsing the SMILES string. The output of the preceding program is the following:

```
Number of benzene atoms: 6
Number of phenol atoms: 7
```

Reusing a molecule (OEParseSmiles)

```
using System;
using OpenEye.OEChem;

public class ReuseMolecule
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();

        OEChem.OEParseSmiles(mol, "c1ccccc1");
        Console.WriteLine("Number of benzene atoms: " + mol.NumAtoms());

        OEChem.OEParseSmiles(mol, "c1ccccc1O");
```

(continues on next page)

(continued from previous page)

```
        Console.WriteLine("Number of phenol atoms: " + mol.NumAtoms());
    }
}
```

In the second example the **low-level** *OEParseSmiles* function is called. The output of the preceding program is the following:

```
Number of benzene atoms: 6
Number of phenol atoms: 13
```

The second line, `Number of phenol atoms: 13`, will be surprising to some. The behavior of the *OEParseSmiles* function is to **add** the given SMILES to the current molecule. **OEChem TK** provides a mechanism for reusing a molecule by calling the *Clear* method. *Clear* deletes all atoms and bonds of a molecule, thereby resetting a molecule into its original "empty" state.

Clearing and reusing a molecule (OEParseSmiles)

```
using System;
using OpenEye.OEChem;

public class ReuseMoleculeClear
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();

        OEChem.OEParseSmiles(mol, "c1ccccc1");
        Console.WriteLine("Number of benzene atoms: " + mol.NumAtoms());

        mol.Clear();

        OEChem.OEParseSmiles(mol, "c1ccccc10");
        Console.WriteLine("Number of phenol atoms: " + mol.NumAtoms());
    }
}
```

The output of the preceding program is the following

```
Number of benzene atoms: 6
Number of phenol atoms: 7
```

Using the *Clear* method is recommended, for example, when processing multiple molecules sequentially in a database. Instead of requiring a new molecule to be allocated and destroyed for each entry, the *Clear* method can be used to reset a molecule to its initial "empty" state.

Unique Representation

It is sometimes useful to generate a unique representation of a molecule for use as an identifier for a database key. The compact nature of SMILES strings makes them ideal candidates for the task. However, the same molecule can be represented by many different SMILES strings. **OEChem TK** features an advanced algorithm for generating a (unique) canonical isomeric SMILES string. A canonical isomeric SMILES string can be generated from a molecule by calling the *OEMolToSmiles* or *OEMolToCXSMILES* functions.

Creating a canonical isomeric SMILES string from a molecule

```
using System;
using OpenEye.OEChem;

public class OEMolToSmiles
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "C1=CC=CC=C1");

        string isosmi = OEChem.OEMolToSmiles(mol);
        Console.WriteLine("Canonical isomeric SMILES is " + isosmi);
    }
}
```

The output of the preceding program is the following:

```
Canonical isomeric SMILES is c1ccccc1
```

The following slightly more complicated example reads SMILES from standard input and writes the corresponding canonical isomeric SMILES to standard output.

Creating canonical isomeric SMILES strings

```
using System;
using System.IO;
using OpenEye.OEChem;

public class OEMolToSmilesRW
{
    public static int Main(string[] args)
    {
        string str;
        try
        {
            StreamReader isr = new StreamReader(Console.OpenStandardInput());

            while ((str = isr.ReadLine()) != null)
            {
                OEGraphMol mol = new OEGraphMol();
                if (OEChem.OESmilesToMol(mol, str))
                {
                    string isosmi = OEChem.OEMolToSmiles(mol);
                    Console.WriteLine(isosmi);
                }
                else
                {
                    OEChem.OEThrow.Warning(str + " is an invalid SMILES!");
                }
            }
        }
        catch (Exception e)
```

(continues on next page)

(continued from previous page)

```
        {
            Console.WriteLine(e);
        }
        return 0;
    }
}
```

input	output (canonical isomeric SMILES)
C1CCCN[C@@H]1(O)	C1CCN[C@@H](C1)O
C1CN[C@H](O)CC1	C1CCN[C@@H](C1)O
C1CC[C@H](O)CC1	C1CC[C@@H](CC1)O
C1CCC(O)CC1	C1CCC(CC1)O
C1=NC=CN1C[C@H](N)C(=O)O	c1cn(cn1)C[C@@H](C(=O)O)N

The *OEMolToSmiles* and *OEMolToCXSMILES* functions are considered a **high-level** functions. Prior to creating the canonical isomeric SMILES, the *OEMolToSmiles* function perceives the following properties *if necessary*:

- the rings of the molecule, by using *OEFindRingAtomsAndBonds*.
- the aromaticity of the molecule, by calling the *OEAssignAromaticFlags*. function using the *OEAroModel.OpenEye* aromaticity model
- the atom and bond stereochemistry.

It is also possible to generate canonical SMILES without isomeric information by using the *OECreateCanSmiString* **low-level** function. As was shown in the *Construction from SMILES* section, *OEParseSmiles* preserves the aromaticity present in the input SMILES string. The function *OEAssignAromaticFlags* has to be used to perceive aromaticity in a molecule.

Creating canonical SMILES strings

```
using System;
using System.IO;
using OpenEye.OEChem;

public class CreateSMILESFromOEGraphMolRW
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        string str;
        try
        {
            StreamReader isr = new StreamReader(Console.OpenStandardInput());

            while ((str = isr.ReadLine()) != null)
            {
                mol.Clear();
                if (OEChem.OEParseSmiles(mol, str))
                {
                    OEChem.OEAssignAromaticFlags(mol);
                    string cansmi = OEChem.OECreateCanSmiString(mol);
```

(continues on next page)

(continued from previous page)

```
                    Console.WriteLine(cansmi);
                }
                else
                {
                    OEChem.OEThrow.Warning(str + " is an invalid SMILES!");
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        return 0;
    }
}
```

Notice that the preceding program does not construct and destruct molecules each time through the loop, but rather uses the *Clear* function to reuse the molecule. If the line `mol.Clear()` were removed from the program, the output would contain longer and longer SMILES containing disconnected fragments. See section *Reuse* for more details.

input	output (canonical SMILES)
c1cccnc1(O)	c1ccnc(c1)O
C1=CC=CC=C1	c1ccccc1
C1=CN=CC=C1	c1ccncc1
C1=CC=CC=N1	c1ccncc1
C1=NC=CN1CCC(=O)O	c1cn(cn1)CCC(=O)O

Hint

We highly recommend the usage of the *OEMolToSmiles* function when creating a SMILES string.

We highly recommend the usage of the *OEMolToCXSMILES* function when structures may contain enhanced stereogroup information.

See also

- *OEMolToSmiles*
- *OEMolToCXSMILES*

InChI

Canonical SMILES is not the only unique representation available. The IUPAC International Chemical Identifier (InChI), and its corresponding hashkey representation (InChIKey) are also unique to the compound they describe [InChI-2013]. InChIs can be created from molecules using the *OECreateInChI*, *OEMolToInChI*, or *OEMolToSTDInChI* functions.

Creating standard InChI

```
using System;
using OpenEye.OEChem;

public class ParseCreateStandardInChI
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccnc(c1)O");

        string inchi = OEChem.OEMolToSTDInChI(mol);
        Console.WriteLine("InChI is " + inchi);
    }
}
```

input (SMILES)	output (Standard InChI)
c1ccnc(c1)O	InChI=1S/C5H5NO/c7-5-3-1-2-4-6-5/h1-4H,(H,6,7)

Note

The 'S' in 'InChI=1S' denotes standard InChI.

The following slightly more complicated example reads InChI strings from standard input and writes InChI strings to standard output.

Reading and writing InChI strings

```
using System;
using System.IO;
using OpenEye.OEChem;

public class OEMolToInChIRW
{
    public static int Main(string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        string str;
        try
        {
            StreamReader isr = new StreamReader(Console.OpenStandardInput());

            while ((str = isr.ReadLine()) != null)
            {
                OEGraphMol mol = new OEGraphMol();
                if (OEChem.OEInChIToMol(mol, str))
                {
                    string inchi = OEChem.OEMolToSTDInChI(mol);
                    Console.WriteLine(inchi);
                }
                else
                {
                    OEChem.OEThrow.Warning(str + " is an invalid InChI!");
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        return 0;
    }
}
```

A nonstandard InChI can be generated by passing in an *OEInChIOptions* object to the *OECreateInChI* function. The options available are documented in the *OEInChIOptions* class.

Creating nonstandard InChI strings

```
using System;
using OpenEye.OEChem;

public class CreateNonStandardInChI
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccnc(c1)O");

        OEInChIOptions opts  = new OEInChIOptions();
        opts.SetFixedHLayer(true);
        string inchi = OEChem.OECreateInChI(mol,opts);
        Console.WriteLine("InChI is " + inchi);
    }
}
```

input (SMILES)	output (nonstandard InChI)
c1ccnc(c1)O	InChI=1/C5H5NO/c7-5-3-1-2-4-6-5/h1-4H,(H,6,7)/f/h7H

The 27-character-long InChIKey is made of three parts connected by hyphens. The first part is 14 characters long and is based on the connectivity and proton layers of an InChI string. The second part, contains 9 characters that are related to all other InChI layers (isotopes, stereochemistry, etc.) and also contains the version of InChI and its standard/nonstandard property in the last two characters. The third part is one letter, describing the (de)protonation layer of the original InChI.

CreateInChIKey

```
using System;
using OpenEye.OEChem;

public class CreateInChIKey
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccnc(c1)O");

        string inchikey = OEChem.OEMolToSTDInChIKey(mol);
        Console.WriteLine("InChIKey is " + inchikey);
    }
}
```

input (**SMILES**)	output (InChI Key)
c1ccnc(c1)O	UBQKCCHYAOITMY-UHFFFAOYSA-N

2.1.2 Molecule Reading and Writing

Molecule Streams

The *previous section* demonstrated reading and writing *SMILES* strings from standard input. This required the programmer to perform the file I/O explicitly. Whilst this may be reasonable for *SMILES* strings that may be read via STD getline, it isn't suitable for more complex file formats. To ease this task, **OEChem TK** provides the *oemolstream* abstraction. The classes *oemolistream* and *oemolostream* allow input and output of molecules using the *OEReadMolecule* and *OEWriteMolecule* functions, respectively.

Listing 1: High-level Molecule I/O using molstreams

```
using System;
using OpenEye.OEChem;

public class oemolstream
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream();
        oemolostream ofs = new oemolostream();

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

Listing 1 will read molecules from stdin in *SMILES* format and write them to stdout in *absolute SMILES* format. Notice that in this example, the following is done automatically for you by *OEReadMolecule*:

- *OEMolBase.Clear* method to reset the molecule.
- *OEAssignAromaticFlags* to normalize aromaticity.
- *OEMolBase.operator bool* to test the validity of the molecule.

Note

OEReadMolecule will skip any invalid molecules automatically.

Since the same *OEGraphMol* instance is being reused for every molecule in the stream, special precautions must be taken to load up an entire file into memory. *Listing 2* will read molecules into memory by making sure there is a copy made when inserting the molecule into the container. This is useful if the molecules need to be repeatedly accessed.

Listing 2: Reading molecules into memory

```
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class ReadMolIntoMemory
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream();
        List<OEGraphMol> mollist = new List<OEGraphMol>();

        // read molecules and put copies into list
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            Console.WriteLine(mol.GetTitle());
            mollist.Add(new OEGraphMol(mol));
        }

        Console.WriteLine(mollist.Count.ToString() + " molecules loaded.");
    }
}
```

File Formats

In addition to *SMILES* strings, **OEChem TK** is able to read numerous other molecular file formats, including *MDL SD* files, Tripos *Tripos MOL2* files and *PDB* files. The format of an input file or stream may be associated with an *oemolistream* using the *SetFormat* method, and may be retrieved with *GetFormat*. The file format is represented by an unsigned integer, which should be one of the constants defined in the *the OEFormat namespace*. A value of *OEFormat.UNDEFINED* means that there is no file format associated with the *oemolstream*.

> **Note**
>
> The default format associated with an *oemolstream* is *OEFormat.SMI*.

Listing 3 demonstrates how to use *oemolstreams* to convert *MDL SD* files into *Tripos MOL2* files.

Listing 3: Explicitly setting file formats

```
using System;
using OpenEye.OEChem;

public class ReadWriteSettingFormats
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream();
        oemolostream ofs = new oemolostream();

        ifs.SetFormat(OEFormat.SDF);
        ofs.SetFormat(OEFormat.MOL2);

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

Table 1: **OEChem TK file format support**

File Format	Description	Read	Write
OEFormat.CAN	Canonical SMILES	Yes	Yes
OEFormat.CDX	ChemDraw CDX	Yes	Yes
OEFormat.CIF	Crystallographic Information File	Yes	Yes
OEFormat.CSV	Comma separated values	Yes	Yes
OEFormat.CXSMILES	Chemaxon Extended SMILES	Yes	Yes
OEFormat.FASTA	FASTA protein sequence	Yes	Yes
OEFormat.INCHI	IUPAC InChI	No	Yes
OEFormat.INCHIKEY	IUPAC InChI Key	No	Yes
OEFormat.ISM	Canonical isomeric SMILES	Yes	Yes
OEFormat.MDL	MDL Mol File	Yes	Yes
OEFormat.MF	Molecular Formula (Hill order)	No	Yes
OEFormat.MMCIF	Macromolecular Crystallographic Information File	Yes	Yes
OEFormat.MMOD	Macromodel	Yes	Yes
OEFormat.MOL2	Tripos Sybyl mol2 file	Yes	Yes
OEFormat.MOL2H	Sybyl mol2 with explicit hydrogens	Yes	Yes
OEFormat.MOPAC	MOPAC file format(s)	No	Yes
OEFormat.OEB	*OpenEye* OEBinary	Yes	Yes
OEFormat.OEZ	Zstd Compressed *OpenEye* OEBinary	Yes	Yes
OEFormat.PDB	Protein Databank PDB file	Yes	Yes
OEFormat.RDF	MDL RDF File	Yes	No
OEFormat.SDF	MDL SD File	Yes	Yes
OEFormat.SKC	MDL ISIS Sketch File	Yes	No
OEFormat.SLN	Tripos Sybyl Line Notation	No	Yes
OEFormat.SMI	Canonical isomeric SMILES	Yes	Yes
OEFormat.USM	SMILES	Yes	Yes
OEFormat.XYZ	XMol XYZ format	Yes	Yes

File Input and Output

In addition to standard in and standard out, **OEChem TK**'s *oemolstreams* also support reading from files. To open a file, use the *open* method with the required filename. For input *oemolistreams*, if the file does not exist the *open* fails and returns false, for output *oemolostreams*, the output file is created if it did not previously exist and overwritten if it did. If a filename is passed to the *open* method, the format and gz compression flags are set based on the file extension of the filename for the *oemolistream* or *oemolostream* instance. If *no* arguments are passed to *open*, an *oemolistream* will use standard in, and an *oemolostream* will use standard out, the format will be set to *OEFormat.Default* and gz compression will be disabled. Much like regular file I/O, *oemolstreams* can be closed after use, using the *close* method. *oemolstreams* are automatically closed by their destructors.

Listing 4: Reading and writing molecule files

```
using System;
using OpenEye.OEChem;

public class ReadWriteToFiles
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream();
        oemolostream ofs = new oemolostream();
```

(continues on next page)

(continued from previous page)

```
        if (!ifs.open("input.sdf"))
        {
            OEChem.OEThrow.Fatal("Unable to open 'input.sdf'");
        }

        if (!ofs.open("output.mol2"))
        {
            OEChem.OEThrow.Fatal("Unable to create 'output.mol2'");
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

One convenient feature of the *open* method of *oemolstreams* is that it sets the file format and gz compression associated with the stream from the file extension of the filename used as an argument. *Listing 4* converts the file 'input.sdf' in MDL file format into the file 'output.mol2' in Tripos MOL2 file format. This behavior can be overridden by calling *SetFormat* after the *open* but before the first molecule is read or written to a stream.

String Input and Output

The *oemolistream* and *oemolostream* provide the ability to read and write memory buffers instead of files.

This is accomplished through the methods, *oemolistream.OpenByteArray* and *oemolostream.GetByteArray*.

The *Listing 5* example below demonstrates how to read and write molecules from memory buffers.

Listing 5: Reading and writing molecule from memory buffers

```
using System;
using System.IO;
using System.Text;
using OpenEye.OEChem;

public class StringIO
{
    public static void Main(string[] argv)
    {
        byte[] molbarray = System.Text.Encoding.UTF8.GetBytes("c1cnccc1");
        OEGraphMol mol = new OEGraphMol();
        ReadFromByteArray(molbarray, mol);

        byte [] molbarraySDF = WriteToByteArray(mol);
        string molstring = System.Text.Encoding.UTF8.GetString(molbarraySDF, 0,␣
↪molbarraySDF.Length);
        Console.WriteLine("MOL string");
        Console.WriteLine(molstring);
    }
```

(continues on next page)

(continued from previous page)

```
    private static byte[] WriteToByteArray(OEMolBase mol)
    {
        oemolostream ofs = new oemolostream();
        ofs.SetFormat(OEFormat.SDF);
        ofs.openstring();
        OEChem.OEWriteMolecule(ofs, mol);
        return ofs.GetByteArray();
    }

    private static void ReadFromByteArray(byte[] molArray, OEGraphMol mol)
    {
        oemolistream ifs = new oemolistream();
        ifs.SetFormat(OEFormat.SMI);
        ifs.OpenByteArray(molArray, (uint)molArray.Length);
        OEChem.OEReadMolecule(ifs, mol);
    }
}
```

Compressed Input and Output

For any of the molecular file formats supported by **OEChem TK**, it is often convenient to read and write compressed *files* or *strings*. Molecule streams support gzipped input and output via the zlib library. The `.gz` suffix on any filename used to open a stream is recognized and the stream is read or written in compressed format. This mechanism does not interfere with the format perception. For instance, `foo.sdf.gz` is recognized as a gzipped file in MDL's SD format.

The *Listing 6* example below converts all of the molecules in a gzipped SD format file into a gzipped OEBinary file.

Listing 6: Reading and writing compressed molecule files

```
using System;
using OpenEye.OEChem;

public class ReadWriteCompressedFiles
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream("input.sdf.gz");
        oemolostream ofs = new oemolostream("output.oeb.gz");

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

The **OEChem TK**'s binary file format *OEFormat.OEB* is well suited (and our preferred way) to store multi-conformer molecules. When working with multi-conformer molecules of +1M datasets, the generated files can easily reach a size when transferring them between computers or networks can be slow. One way to handle this issue is to compress the files with gzip before transfer. As the above example shows **OEChem TK** seamlessly handles gzipped molecule

files with gz extension. Alternatively, **OEChem TK** also provides a native compressed file format, called *OEFormat.OEZ*, with the oez file extension. This file format currently only supports multi-conformer molecules derived from the *OEMCMolBase* class.

One of the main differences between the gzipped *OEFormat.OEB* and the *OEFormat.OEZ* files that while gzip compresses a molecular file as a whole, in an *OEFormat.OEZ* file, each molecule is individually compressed using the Zstandard compression algorithm.

This difference makes *OEFormat.OEZ* suitable to work really well with *OEMolDatabase* that is designed to provide fast read-only random access to molecular files. When an *OEMolDatabase* opens a gzipped molecule file, uncompressing the whole file is necessary to access any molecule in the dataset. When working with a *OEFormat.OEZ* file, accessing a specific molecule in the file only triggers the un-compression of the associated data chunk.

The table below lists the file sizes of the same dataset of 1M molecules in uncompressed and compressed formats with different numbers of conformations per molecule.

Dataset	oeb	oeb.gz	oez
1M molecules with 10 conformations	5.4GB	1.9GB	2.2GB
1M molecules with 50 conformations	22GB	5.7GB	5.5GB
1M molecules with 200 conformations	69GB	16GB	13GB

Note

The *OEFormat.OEZ* file format was introduced in the 2019.Apr toolkits. OpenEye applications or toolkits released before Apr/2019 will **not** be able to read or write this new file format.

Command Line Format Control

Using the methods outlined above, it is possible to allow the stream format to be controlled from the command line. **OEChem TK**'s *oemolstreams* control the format by interpreting the input and output file names.

Listing 7: Controlling File Format from the Command Line

```
using System;
using OpenEye.OEChem;

public class ReadWriteCommandLineArgs
{
    public static int Main(string[] args)
    {
        if (args.Length != 2)
        {
            OEChem.OEThrow.Usage("ReadWriteCommandLineArgs <input> <output>");
        }

        oemolistream ifs = new oemolistream();
        oemolostream ofs = new oemolostream();

        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0]);
        }
```

(continues on next page)

(continued from previous page)

```
        if (!ofs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to create " + args[1]);
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
        }
        return 0;
    }
}
```

Listing 7 is an example of using command-line arguments to allow **OEChem TK** programs to support many file formats at run-time. For instance, if *Listing 7* is a program called *convert*:

```
prompt> convert file1.sdf file1.smi
```

This command will convert *file1.sdf* from SD format to SMILES format. A first extension of this idea allows access to standard in and standard out via the "-" filename. For instance:

```
prompt> convert file2.mol2 -
```

This command will convert `file2.mol2` in MOL2 format and write the molecules to standard out in SMILES, the default format.

Thus if you have another program, *GetFromDatabase*, which retrieves molecules from a database and writes them in SMILES format, you can chain it with any **OEChem TK** program. Using your operating systems redirection commands (*e.g.* - Unix pipe "|" or redirect ">") you can move molecules directly from *GetFromDatabase* to *convert* without a temporary file:

```
prompt> GetFromDatabase | convert - file3.sdf
```

This command will take the SMILES format output from *GetFromDatabase* and generate an SD format file.

However, to make this concept of using standard in and standard out for piping data really useful, one needs to be able to control the format of standard in and standard out similarly to the way it would be controlled for temporary files. To facilitate this, *oemolstreams* interpret filenames which are ONLY format extensions to indicate format control for standard in and standard out.

Listing 8: Controlling standard in and standard out File Format

```
using System;
using OpenEye.OEChem;

public class ReadWriteStdinStdoutFormatC
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream(".sdf");
        oemolostream ofs = new oemolostream(".mol2");
```

(continues on next page)

(continued from previous page)

```
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

Listing 8 is an example of opening an *oemolistream* to standard in with the format set to SDF. The output *oemolostream* is opened to standard out with the format set to MOL2. This is exactly equivalent to *Listing 3*. However, it demonstrates the extensibility of controlling the format of standard in and standard out from the command line.

Note

This prevents you from naming files as pure extensions: `.mol2`, `.sdf`, etc.

Now, using our program *convert* from *Listing 7*

it is possible to do the following:

```
prompt> convert .sdf .mol2
```

This command opens standard in with the SDF format and opens standard out with the MOL2 format.

Now there is complete format control of standard in and standard out from the command line. If we have a program *GenerateStructures*, which only writes the MOL2 format and another program *GenerateData*, which only reads the SD format, we can use them from the command line with any **OEChem TK** program which uses command-line arguments for file specification:

```
prompt> GenerateStructures | convert .mol2 .sdf | GenerateData
```

This command demonstrates how any **OEChem TK** program with command-line file specification can be used to pipe formatted input and output.

Flavored Input and Output

The general goal of the *oemolstream* input and output classes in **OEChem TK** is to provide the user with transparent access to the very complex task of reading and writing molecules in a wide variety of formats. However, occasionally, a programmer may want to tweak the behavior of specific readers or writers without abandoning the *oemolstreams* to use the low level writers (such as *OEWriteMDLFile*). For these instances, *oemolstreams* provide the *oemolstreambase.SetFlavor* and *oemolstreambase.GetFlavor* methods.

The *oemolstreambase.SetFlavor* method takes two unsigned integer arguments, the first is the format for which the flavor is being specified and the second is the flavor itself. The formats are specified as discussed in *File Formats*. The input flavors are specified in the *OEIFlavor* namespace and the output flavors are specified in the *OEOFlavor* namespace. Unlike the formats, the flavors are a bitmask and may be binary OR'd together. Under the *OEIFlavor* and *OEOFlavor* namespaces, there is a namespace for each format as well as a *OEIFlavor.Generic* namespace. The *OEOFlavor.Generic* namespace is used to control aromaticity perception and other properties common to all formats. To completely specify a flavor, one would typically binary-OR a *OEOFlavor.Generic* flag and a format specific flag and pass the resultant value to *oemolstreambase.SetFlavor*.

The default behavior for the *PDB reader* is that *TER* specifies the termination of a disconnected fragment within the same molecule while *END* specifies the termination of a connection table. However, some users may want to have the reader split *PDB* input files into different molecules every time a *TER* appears.

The following code is an example of changing the *PDB* reader flavor.

Listing 9: Changing oemolistream Reader Flavor

```
using System;
using OpenEye.OEChem;

public class ReadWriteFlavors
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream("input.pdb");
        oemolostream ofs = new oemolostream("output.mol2");
        uint flavor =
          OEIFlavor.Generic.Default | OEIFlavor.PDB.Default | OEIFlavor.PDB.TER;
        ifs.SetFlavor(OEFormat.PDB, flavor);

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

Similar low-level control can be exerted over both input and output *oemolstreams* using the powerful *oemolstreambase.SetFlavor* method. See the API documentation for the associated low-level routine and namespace for details on the effects of specific flavor flags.

Table 2: **OEChem TK low-level readers and input flavors**

File Format	Low-Level Reader	Input Flavors
OEFormat.CAN	*OEParseSmiles*	*OEIFlavor.CAN*
OEFormat.CDX	*OEReadCDXFile*	*OEIFlavor.CDX*
OEFormat.CIF	*OEReadCIFFile*	*OEIFlavor.CIF*
OEFormat.CSV	*OEReadCSVFile*	*OEIFlavor.CSV*
OEFormat.CXSMILES		*OEIFlavor.CXSMILES*
OEFormat.FASTA	*OEReadFASTAFile*	*OEIFlavor.FASTA*
OEFormat.ISM	*OEParseSmiles*	*OEIFlavor.ISM*
OEFormat.MDL	*OEReadMDLFile*	*OEIFlavor.MDL*
OEFormat.MF		
OEFormat.MMCIF	*OEReadCIFFile*	*OEIFlavor.MMCIF*
OEFormat.MMOD	*OEReadMacroModelFile*	*OEIFlavor.MMOD*
OEFormat.MOL2	*OEReadMol2File*	*OEIFlavor.MOL2*
OEFormat.MOL2H	*OEReadMol2File*	*OEIFlavor.MOL2H*
OEFormat.MOPAC		
OEFormat.OEB	*OEReadOEBFile*	
OEFormat.OEZ	N/A (*OEReadMolecule* works)	
OEFormat.PDB	*OEReadPDBFile*	*OEIFlavor.PDB*
OEFormat.SDF	*OEReadMDLFile*	*OEIFlavor.SDF*
OEFormat.SKC	*OEReadSketchFile*	*OEIFlavor.SKC*
OEFormat.SLN		
OEFormat.SMI	*OEParseSmiles*	*OEIFlavor.SMI*
OEFormat.XYZ	*OEReadXYZFile*	*OEIFlavor.XYZ*

Table 3: **OEChem TK low-level writers and output flavors**

File Format	Low-Level Writer	Output Flavors
OEFormat.CAN	*OECreateCanSmiString*	*OEOFlavor.CAN*
OEFormat.CIF	*OEWriteCIFFile*	*OEOFlavor.CIF*
OEFormat.CDX	*OEWriteCDXFile*	*OEOFlavor.CDX*
OEFormat.CSV	*OEWriteCSVFile*	*OEOFlavor.CSV*
OEFormat.CXSMILES		*OEOFlavor.CXSMILES*
OEFormat.FASTA	*OEWriteFASTAFile*	*OEOFlavor.FASTA*
OEFormat.ISM	*OECreateIsoSmiString*	*OEOFlavor.ISM*
OEFormat.MDL	*OEWriteMDLFile*	*OEOFlavor.MDL*
OEFormat.MF	*OEMolecularFormula*	*OEOFlavor.MF*
OEFormat.MMCIF	*OEWriteCIFFile*	*OEOFlavor.MMCIF*
OEFormat.MMOD	*OEWriteMacroModelFile*	*OEOFlavor.MMOD*
OEFormat.MOL2	*OEWriteMol2File*	*OEOFlavor.MOL2*
OEFormat.MOL2H	*OEWriteMol2File*	*OEOFlavor.MOL2H*
OEFormat.MOPAC	*OEWriteMOPACInputFile*	*OEOFlavor.MOPAC*
OEFormat.OEB	N/A (*OEWriteMolecule* works)	
OEFormat.OEZ	N/A (*OEWriteMolecule* works)	
OEFormat.PDB	*OEWritePDBFile*	*OEOFlavor.PDB*
OEFormat.SDF	*OEWriteMDLFile*	*OEOFlavor.SDF*
OEFormat.SKC		
OEFormat.SLN	*OECreateSlnString*	*OEOFlavor.SLN*
OEFormat.SMI	*OECreateSmiString*	*OEOFlavor.SMI*
OEFormat.XYZ	*OEWriteXYZFile*	*OEOFlavor.XYZ*
OEFormat.INCHI	*OECreateInChI*	*OEOFlavor.INCHI*
OEFormat.INCHIKEY	*OECreateInChIKey*	*OEOFlavor.INCHIKEY*

Note

Low-level readers will not call *Clear* on the molecule before reading the molecule into it. This is useful for concatenating multiple molecules together.

Molecular Property Preservation

> **Warning**
>
> *OEWriteMolecule* will change your molecule!

The high-level *OEWriteMolecule* function standardizes the molecule according to the output type for uniformity. For writing molecules without changing them, there are two options. If you would like the data to appear in the file exactly as it is in the molecule (perhaps *Tripos* atom names in a .pdb format), then you should use a low level writer. On the other hand, if you would like to write a standardized molecule (*e.g. Tripos* atom types in a MOL2 file), but don't want your molecule changed, you can use *OEWriteConstMolecule*. The *OEWriteConstMolecule* function makes a copy of the input molecule before performing the necessary perceptions and standardizations.

2.1.3 Tagged Data Manipulation

SD Tagged Data Manipulation

Meta information about a molecule is stored in what is known as 'tagged data'. The most common example of this is the data fields found in SDF files. Since SD files are a common form of data storage and transfer from one system to another, **OEChem TK** provides several methods to manipulate this data. **OEChem TK** also provides round tripping SD data through the OEBinary, `.oeb`, and comma-separated-values, `.csv` file formats.

A simple class, *OESDDataPair* is used to set and retrieve SD data.

Data	Set method	Get method
tag	*OESDDataPair.SetTag*	*OESDDataPair.GetTag*
data	*OESDDataPair.SetValue*	*OESDDataPair.GetValue*

The following functions provide access to the SD data.

Table 4: **Functions to manipulate SD data**

Function	Description
OEIsSDDataFormat	determines whether a file format support SD data
OESetSDData	set a tag and value data pair
OEAddSDData	add a tag and value data pair
OEHasSDData	determine whether a molecule has data with a given tag
OEGetSDData	get the value for the given tag
OEGetSDDataPairs	return an iterator over all the SD data pairs of the molecule
OECopySDData	copy the entire set of SD data from one molecule to another
OEDeleteSDData	delete all SD data items with the given tag
OEClearSDData	clear all SD data from a molecule

Since *OESDDataPair* stores SD data information in a string, numeric values have to be converted to a string with C#'s `ToString()` method before storing a value with either *OESetSDData* or *OEAddSDData* functions.

```
OEChem.OESetSDData(mol, "number of atoms", mol.NumAtoms().ToString());
```

Similarly the retrieved string can be converted to a numeric value with `ToSingle()` style methods.

```
if (OEChem.OEHasSDData(mol, "weight"))
{
    string str = OEChem.OEGetSDData(mol, "weight");
```

(continues on next page)

(continued from previous page)

```
    float weight = System.Convert.ToSingle(str);
    System.Console.WriteLine("weight= " + weight);
}
```

The following example shows how to manipulate SD tagged data.

Listing 1: SD data manipulation

```
using System;
using OpenEye.OEChem;

public class ManipulateSDData
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccccc1");
        mol.SetTitle("benzene");

        // set some tagged data
        OEChem.OESetSDData(mol, "color", "brown");
        OEChem.OESetSDData(mol, new OESDDataPair("size", "small"));
        DumpSDData(mol);

        // check for existence of data, then delete it
        if (OEChem.OEHasSDData(mol, "size"))
        {
            OEChem.OEDeleteSDData(mol, "size");
        }

        DumpSDData(mol);

        // add additional color data
        OEChem.OEAddSDData(mol, "color", "black");
        DumpSDData(mol);

        // remove all SD data
        OEChem.OEClearSDData(mol);
        DumpSDData(mol);

        return 0;
    }

    private static void DumpSDData(OEMolBase mol)
    {
        System.Console.WriteLine("SD data of " + mol.GetTitle());

        // loop over SD data
        foreach (OESDDataPair dp in OEChem.OEGetSDDataPairs(mol))
        {
            System.Console.WriteLine(dp.GetTag() + " : " + dp.GetValue());
        }
```

(continues on next page)

(continued from previous page)

```
        System.Console.WriteLine();
    }
}
```

Note

If a data with the same tag already exists:

- it is replaced, when using *OESetSDData* function
- it is added, when using *OEAddSDData* function

The output of the preceding program is the following:

```
SD data of benzene
color : brown
size : small

SD data of benzene
color : brown

SD data of benzene
color : brown
color : black

SD data of benzene
```

Note

Note that SD tagged data is specific to MDL's SD file format. Any SD data added to a molecule will only be written out to SD files, OEBinary files, or CSV files. The SD data fields will only be filled when reading from SD files that contain SD tagged data, CSV data files, or from OEBinary files previously created to contain this data.

See also

- *Generic Data* chapter

Multi-conformer molecules

See also

For using tag data with multi-conformer molecules, see *Dude, where's my SD data?*.

CSV File Format

The CSV, comma-separated-values, file format is a comma file format used for data exchange in a wide variety of software. Over the years, the popularity of the format has out-stripped the robustness of many of its implementations leading to some subtle differences in how it is implemented between different software packages. This has led to

creation and adoption of RFC 4180 as a standard for the CSV format. **OEChem TK** supports this standard as the base format for reading and writing molecules, *OEMolBase* objects, to `.csv` files. **OEChem TK** will read and write `.csv` files in the following layout:

1. The first line can optionally contain a header to use as the SD tag.
   - **OEChem TK** decides if the first line is a header by trying to interpret the first column of the first line as SMILES. See the *OEIFlavor.CSV.Header* flavor.
2. The first column is expected to be a SMILES string representing the molecule.
3. The second column is the molecule title, as accessed through *OEMolBase.GetTitle*.
4. Remaining columns will be read into SD data fields. The data is then accessible through *OEGetSDData* and *OESetSDData*.
   - If the file did not contain a header line, the SD tags will be named in the following pattern: `OE_CSV_COLUMN_1`, `OE_CSV_COLUMN_2`, ..., `OE_CSV_COLUMN_N`.
5. New-line characters are not supported inside a CSV record. They are round-tripped using the pipe character, `|`, instead.

Point #5 is a subtlety of meta-data interchange between SD files and CSV files. On the one hand, many CSV implementations do not support line-breaks inside CSV fields, Microsoft Excel does not and neither does **OEChem TK**. However, it is common to use multiple lines inside an SD data file like the following:

```
>  <activ_class>
Antiarthritic
Antiinflammatory
Cyclooxygenase-2 Inhibitor
```

OEChem TK implements a solution to this problem that ChemAxon has implemented and documented here. The solution is to use the pipe character, `|`, to represent line breaks inside a data field. For example, the previous example of a multi-line SD data field would look like the following:

```
SMILES,TITLE,active_class
...,...,Antiarthritic|Antiinflammatory|Cyclooxygenase-2 Inhibitor
```

This does cause a different problem for pipe characters inside data fields: how pipe characters are escaped? **OEChem TK** works around this problem by embedding the same semantics for CSV parsing to parse multi-line fields, just using the pipe character as the separator and the single quote character as the quoting escape character. *Listing 2* demonstrates an example that writes out a tricky set of data to the *OEFormat.CSV* format.

Listing 2: Writing out SD data as CSV

```
using System;
using OpenEye.OEChem;

public class WriteCSV
{
    public static void Main(string[] args)
    {
        oemolostream ofs = new oemolostream();
        ofs.SetFormat(OEFormat.CSV);

        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "O");
        mol.SetTitle("water");
```

(continues on next page)

(continued from previous page)

```
        OEChem.OESetSDData(mol, "phases", "gas,liquid,solid");
        OEChem.OESetSDData(mol, "uses", "bathing\nwater guns\ntea|earl grey|hot");

        OEChem.OEWriteMolecule(ofs, mol);

        ofs.close();
    }
}
```

The output of *Listing 2* will look like the following:

```
SMILES,TITLE,phases,uses
O,water,"gas,liquid,solid",bathing|water guns|'tea|earl grey|hot'
```

Code Example

- Reordering CSV File **OpenEye Python Cookbook** recipe

PDB Tagged Data Manipulation

The *OEPDBDataPair* class is used to set and retrieve PDB data pairs.

Data	Set method	Get method
tag	*OEPDBDataPair.SetTag*	*OEPDBDataPair.GetTag*
data	*OEPDBDataPair.SetValue*	*OEPDBDataPair.GetValue*

If you wish to store a numeric value, use C#'s `.ToString()` method to convert it to a string (see examples in *SD Tagged Data Manipulation*).

The following functions provide access to the PDB data.

Table 5: Functions to manipulate PDB data

Function	Description
OESetPDBData	set a tag and value data pair
OEAddPDBData	add a tag and value data pair
OEHasPDBData	determine whether a molecule has data with a given tag
OEGetPDBData	get the value for the given tag
OEGetPDBDataPairs	return an iterator over all the PDB data pairs of the molecule
OECopyPDBData	copy the entire set of PDB data from one molecule to another
OEDeletePDBData	delete all PDB data items with the given tag
OEClearPDBData	clear all PDB data from a molecule

Note

In case of PDB header items like REMARK, each line is treated as a separate instance, Therefore these multiple lines have to be added with *OEAddPDBData* and can be accessed via *OEGetPDBDataPairs*.

The following PDB fields are stored as tagged PDB data when *OEIFlavor.PDB.DATA* input flavor is set:

Table 6: **Imported PDB data fields**

AUTHOR	CAVEAT	COMPND	CRYST1	DBREF
EXPDTA	FORMUL	HEADER	HELIX	HET
HETNAM	HETSYM	JRNL	KEYWDS	MODRES
MTRIX1	MTRIX2	MTRIX3	OBSLTE	ORIGX1
ORIGX2	ORIGX3	REMARK	REVDAT	SCALE1
SCALE2	SCALE3	SEQRES	SEQADV	SHEET
SITE	SOURCE	SPRSDE	SSBOND	TITLE
TURN				

Warning

The tags of PDB data are always 6 characters long and space-padded (for example `"HELIX "` and not `"HELIX"`).

The following example shows how to manipulate PDB tagged data.

Listing 3: PDB data manipulation

```
using System;
using OpenEye.OEChem;

public class ManipulatePDBData
{
    public static int Main(string[] args)
    {
        if (args.Length != 1)
        {
            OEChem.OEThrow.Usage("ManipulatePDBData <pdbfile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0]);
        }

        // need to set input flavor to ensure PDB data is stored on molecule
        ifs.SetFlavor(OEFormat.PDB,
                      OEIFlavor.Generic.Default | OEIFlavor.PDB.Default | OEIFlavor.PDB.
↪DATA);

        OEGraphMol mol = new OEGraphMol();

        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (OEChem.OEHasPDBData(mol, "COMPND"))
            {
                Console.WriteLine("COMPND:");
```

(continues on next page)

(continued from previous page)

```
                Console.WriteLine(OEChem.OEGetPDBData(mol, "COMPND"));
            }

            if (OEChem.OEHasPDBData(mol, "HELIX "))
            {
                Console.WriteLine("HELIX:");
                Console.WriteLine(OEChem.OEGetPDBData(mol, "HELIX "));
            }

            if (OEChem.OEHasPDBData(mol, "SSBOND"))
            {
                Console.WriteLine("SSBOND:");

                foreach (OESDDataPair dp in OEChem.OEGetPDBDataPairs(mol))
                {
                    if (dp.GetTag().Equals("SSBOND"))
                        Console.WriteLine(dp.GetValue());
                }
            }
        }
        return 0;
    }
}
```

The output of the preceding program for 1D1H is the following:

```
COMPND:
    MOL_ID: 1;
HELIX:
   1   1 THR A   11  ASP A   14  5
SSBOND:
   1 CYS A    2    CYS A   16                          1555   1555
   2 CYS A    9    CYS A   21                          1555   1555
   3 CYS A   15    CYS A   28                          1555   1555
```

Note

Note that PDB tagged data is specific to PDB file format. Any PDB data added to a molecule will only be written out to PDB files or OEBinary files.

See also

- *Generic Data* chapter

CIF File Format

The CIF file format is built around keywords, terminology, and structure to allow for a robust and customizable file format for atomic coordinate and biomolecule structure files. This format has generated multiple different sub-genres of CIF file formats: Small Molecule CIF (smCIF), Macro Molecular CIF (mmCIF), Chemical Component CIF (ccCIF), and Structure Factor CIF (sfCIF). All CIF file formats follow the same general terminology. Below is a brief introduction

to this terminology using an mmCIF example, but for a more complete explanation of CIF syntax, see the PDBx/mmCIF User Guide from the wwPDB.

The following example comes from the mmCIF file for the 3TPP PDB code.

```
data_3TPP
#
_entry.id   3TPP
#
_audit_conform.dict_name       mmcif_pdbx.dic
_audit_conform.dict_version    5.397
_audit_conform.dict_location   http://mmcif.pdb.org/dictionaries/ascii/mmcif_pdbx.dic
#
loop_
_database_2.database_id
_database_2.database_code
_database_2.pdbx_database_accession
_database_2.pdbx_DOI
PDB   3TPP         pdb_00003tpp 10.2210/pdb3tpp/pdb
RCSB  RCSB067767   ?            ?
WWPDB D_1000067767 ?            ?
```

Data blocks are used to organize an entire set of related data. Because all the data in this CIF file relate to the 3TPP structure, all data in this example are in the same data block. The start of a data block uses the `data_` keyword at the start of the line.

CIF categories are a grouping of related information within the data block and all data within the same category will have the same category name.

In the above example, there are three categories: `_entity.`, `_audit_conform.`, and `_database_2.`. Toolkit function references to category names require these values to be formatted with a leading underscore + the category name + a trailing period. This category name nomenclature ensures consistency and reduces the potential for misidentifying categories with similar names.

Attributes name the different data types of values saved in the category, and the combination of a category and attribute is called a data item. In the example, the `_database_2.` category contains four attributes: `database_id`, `database_code`, `pdbx_database_accession`, and `pdbx_DOI`, and combining a category and an attribute results in a data item: `_database_2.database_id`.

Categories can come in two types: pair and loop. In a pair category, every data item is paired with a single value. The `_audit_conform.` is a pair type data item. In a loop category, multiple rows of data correlate with the data items where the number of data columns equal and correlate to the number of data items in the category. The `_database_2` is a loop type category.

The *OECIFData* class can be used to access row and attribute (column) data in a category. For example, use the *GetRow("_database_2.", 1)* method to get the second row in the `database_2` category: `"RCSB"`, `"RCSB067767"`, `"?"`, `"?"`. Use the `GetAttributeValues("_database_2.", "database_id")` method to get the `database_id` column values: `"PDB"`, `"RCSB"`, `"WWPDB"`.

See also

- *Generic Data* chapter
- *OECIFData* class
- *OEReadCIFGridFile* function

mmCIF Metadata Manipulation

mmCIF metadata is not stored as tagged data in the same way that PDB data is. Instead, the mmCIF metadata is stored as a single string that can be retrieved, manipulated, and set or appended to.

The following functions provide access to the mmCIF data.

Table 7: Functions to manipulate mmCIF metadata

Function	Description
OESetMMCIFData	Sets mmCIF metadata on a molecule, overwrites if any existing metadata
OEAddMMCIFData	Adds mmCIF metadata on a molecule, appends to any existing metadata
OEHasMMCIFData	Determines whether a molecule has mmCIF metadata stored
OEGetMMCIFData	Retrieves the mmCIF metadata stored on a molecule
OECopyMMCIFData	Copies the mmCIF metadata from one molecule to another
OEClearMMCIFData	Clears the mmCIF metadata from a molecule
OEUpdateMMCIFData	Updates mmCIF metadata using molecule structural information

When writing mmCIF metadata, the default behavior is to write the header as is and without modification so long as it is formatted correctly. The only exception is if important CIF sections of the header are missing, in which case they will be perceived at write time or if header data is updated. See the *mmCIF Header Perception* section below.

For mmCIF, these essential categories are:

- _entity.
- _entity_poly_seq.
- _chem_comp.
- _cell.
- _symmetry.
- _struct_asym.
- _struct_conn.

For Chemical Component CIF, these essential categories are:

- _chem_comp.
- _chem_comp_atom.
- _chem_comp_bond.

To synchronize header data with a molecule, or to manually interact with header data, the *OECIFData* class allows for programmatic access to the header values and includes functionality to update specific sections of the header with a molecule's structural data. The *OEUpdateMMCIFData* function can be used to run a one-time update to the header metadata.

See also

- *Generic Data* chapter
- *OECIFData* class
- *OEUpdateMMCIFData* function

mmCIF Header Perception

A molecule can contain information used to perceive data item values in the mmCIF header. This information is coordinated and can be called using the *OECIFData* class or *OEUpdateMMCIFData* function. Which data items are perceived during that update is controlled by the *OECIFOptions* class. This class breaks perception into seven categories: `Entity`, `ChemComp`, `Cell`, `Symmetry`, `Struct`, `Experimental`, and `Other`. If perception is turned on for any of these categories, category attributes and values can be altered or deleted.

The follow table outlines what CIF categories and attributes could be affected by perception during header updates:

Table 8: OECIFOptions and Corresponding Perceived Categories

OECIFOptions	Related mmCIF Categories
SetPerceiveEntity	**_entity.id** **_entity.type** **_entity_poly.entity_id** **_entity_poly.type** **_entity_poly.pdbx_seq_one_letter_code** **_entity_poly.pdbx_seq_one_letter_code_can** **_entity_poly_seq.entity_id** **_entity_poly_seq.num** **_entity_poly_seq.mon_id** **_entity_poly_seq.hetero**
SetPerceiveChemComp	**_chem_comp.id** **_chem_comp.type** **_chem_comp.mon_nstd_flag** **_chem_comp.name** **_chem_comp.pdbx_synonyms** **_chem_comp.formula** **_chem_comp.formula_weight** **_chem_comp_atom.comp_id** **_chem_comp_atom.atom_id** **_chem_comp_atom.type_symbol** **_chem_comp_atom.charge** **_chem_comp_bond.comp_id** **_chem_comp_bond.atom_id_1** **_chem_comp_bond.atom_id_2** **_chem_comp_bond.type** **_chem_comp_bond.aromaticc**
SetPerceiveStruct	**_struct_asym.id** **_struct_asym.entity_id** **_struct_conn.** _struct. _struct_ref. _struct_ref_seq. _struct_conf. _struct_sheet. _struct_biol. _struct_keywords. _struct_ncs_opr. pdbx_struct_mod_residues.
SetPerceiveSymmetry	_symmetry.entry_id _symmetry.space_group_name_H-M
SetPerceiveCell	_cell. _atom_sites.
SetPerceiveExperimental	_exptl. _refine. _refine_analysis. _refine_is_restr

Categories that are bold are primarily defined or affected by molecule structural data and are consequently entirely redefined when updated. The remaining categories are coordinated with other parts of header data when updating, but primarily contain attributes not perceived by molecular structure. The exhaustive list of these coordinated but not perceived categories are outside the scope of this document, and only the most common are listed.

The `_entity_poly_seq.` category and associated `_entity_poly.` attributes allow for microheterogeneity between the actual molecule sequence and the values in these categories. Consequently, these sequences are not a reflection of the molecule, but a reference to the intended polymeric sequence of molecule. This may result in unexpected behavior during perception if data is missing or inconsistent with the polymeric sequence. When disagreement between attribute values and molecule sequences are encountered, disagreeing values may be deleted from the header. For these sequence-based attributes to persist through perception, all existing values must align with the macro molecules sequence data, otherwise it is considered undefined and will not be populated. If these attribute values are missing prior to perception, they are considered unknowable and will not be populated.

It is important to note that perception and synchronization only occur with the molecule informing header data. Changes in the header will not alter the molecular structure in any way.

Atoms and Bonds

2.1.4 Atom and Bond Traversal

OEChem TK* molecules contain atoms and bonds which have APIs described by the *OEAtomBase* and *OEBondBase* abstract base-classes respectively. Atoms and bonds in **OEChem TK** can **only** be created and destroyed in the context of an **OEChem TK** molecule. While they can be accessed as pointers through various member functions of molecules, their memory is owned by the molecules and they are deallocated during the molecules' destruction. Attempting to use references to atoms or bonds of a molecule after the molecule has gone out of scope results in undefined behavior.

Iterators

The standard way of processing each item or member of a set or collection in **OEChem TK** is by the use of an iterator. The use of iterators is a common abstraction (or design pattern) in object oriented programming because it hides the way the collection/container is implemented from the user. Hence a set of atoms could be implemented internally as an array, a linked list, a hash table, or any similar data structure, but its behavior to the programmer is independent of the actual implementation. An iterator can be thought of as a current position indicator.

OEChem TK makes extensive effort to support the Java iteration syntax.

```
foreach (MyType obj in list)
  obj.DoSomething();
```

So the normal C# user does not have to care that an iterator is providing this convenient, yet powerful, abstraction.

Atom and Bond Iteration

Listing 1 shows the minimal use of **OEChem TK**'s iterators. These examples use the *OEMolBase* methods *GetAtoms* and *GetBonds*, which return iterators over the atoms and bonds of a molecule, respectively.

Listing 1: Using iterators to loop over atoms and bonds

```
using System;
using OpenEye.OEChem;

public class LoopingOverMolAtomsBonds
{
    public static int Main(string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cocc1");

        Console.WriteLine("atoms");
        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            Console.WriteLine(atom.GetAtomicNum());
        }

        Console.WriteLine("bonds");
        foreach (OEBondBase bond in mol.GetBonds())
        {
            Console.WriteLine(bond.GetOrder());
        }
        return 0;
    }
}
```

Note

Listing 1 introduced the *GetAtomicNum* and *GetOrder* methods. These and other *OEAtomBase* and *OEBondBase* methods will be covered in more detail in chapters *Atom Properties* and *Bond Properties*, respectively.

Bonds of an Atom Iteration

The exact same idiom is used for iterating over the bonds attached to an atom. The *GetBonds* method returns an iterator over the bonds connected to that atom. *Listing 2* shows how to use this iterator to determine the explicit degree of an atom, *i.e.* the number of bonds to it, not including bonds to implicit hydrogen atoms.

Listing 2: Looping over the bonds of an atom

```
using System;
using OpenEye.OEChem;

public class LoopingOverBonds
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cocc1Br");

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            Console.WriteLine("Atom " + atom.GetIdx() + " has degree " +␣
↪MyGetExplicitDegree(atom));
        }
        return 0;
    }

    private static int MyGetExplicitDegree(OEAtomBase atom)
    {
```

(continues on next page)

(continued from previous page)

```
        int result = 0;

        foreach (OEBondBase bond in atom.GetBonds())
        {
            result++;
        }

        return result;
    }
}
```

Atom Neighbor Iteration

Often it is not the bonds around the atoms that you wish to loop over, but the neighboring atoms. One way to do this would be to use the *GetBonds* method described in the previous section and use the *GetNbr* method on each *OEBondBase* to get the atom across the bond from the input atom.

Listing 3: Finding the neighbors of an atom (version 1)

```
using System;
using OpenEye.OEChem;

public class LoopingOverNeighbors1
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cocc1Br");

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            Console.Write("Atom: " + atom.GetIdx() + " Neighbors:");

            foreach (OEBondBase bond in atom.GetBonds())
            {
                Console.Write(" " + bond.GetNbr(atom).GetIdx());
            }

            Console.WriteLine();
        }
        return 0;
    }
}
```

However this can be done even more conveniently using the *GetAtoms* method of an *OEAtomBase* directly, which allows loops over the neighbor atoms.

Listing 4: Finding the neighbors of an atom (version 2)

```
using System;
using OpenEye.OEChem;
```

(continues on next page)

(continued from previous page)

```
public class LoopingOverNeighbors2
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cocc1Br");

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            Console.Write("Atom: " + atom.GetIdx() + " Neighbors:");

            foreach (OEAtomBase nbr in atom.GetAtoms())
            {
                Console.Write(" " + nbr.GetIdx());
            }

            Console.WriteLine();
        }
        return 0;
    }
}
```

Atom or Bond Subset Iteration

It can sometimes be useful to loop over a subset of the atoms or bonds of a molecule. Traditionally, this is done with *if* statements inside a loop, but it can sometimes be cleaner and more convenient to subset the members being looped over inside the iterator itself. To do this, many of **OEChem TK**'s iterator generation functions (such as *GetAtoms*) can take an argument which determines which subset of the object to loop over (these functions are called *functors* are detailed in the chapter *Predicate Functors*). The details of these functions are not important here. Instead, a programmer can simply use the predefined functors to control their loops. *Listing 5* shows the use of the predicate *OEHasAtomicNum* to loop over only carbon atoms in a molecule.

Listing 5: Looping over carbon atoms only

```
using System;
using OpenEye.OEChem;

public class LoopingOverCarbon
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1c(Br)occ1CCC");

        Console.Write("Carbon atoms:");
        foreach (OEAtomBase atom in mol.GetAtoms(new OEHasAtomicNum(OEElemNo.C)))
        {
            Console.Write(" " + atom.GetIdx());
        }

        Console.WriteLine();
        return 0;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

See also

For a complete list of built-in predicates, see *Built-in Functors* section.

Iterator Methods

Iterators offer a much wider range of iteration possibilities. For example, the iterator can be reused by using the `ToFirst` method. Or, the order of iteration can be rearranged with the `Sort` method.

The following table describes how to use the same rich set of iterator operations that C++ offers.

Description	C++	C#
Increment	`++i`	`i.Increment()`
Increment by n	`i += n`	`i.Increment(n)`
Decrement	`--i`	`i.Decrement()`
Decrement by n	`i -= n`	`i.Decrement(n)`
Go to first	`i.ToFirst()`	`i.ToFirst()`
Go to last	`i.ToLast()`	`i.ToLast()`
Current Access	`operator->`	`i.Target()`
Validity	`operator bool`	`i.IsValid()`

`Listing 6` shows how to use an *OEAtomBase* iterator to loop over the atoms in a molecule in reverse order and print their atomic numbers.

Note

The order of the atoms returned by `OEMolBase.GetAtoms` can be controlled by `OEMolBase.OrderAtoms`.

Listing 6: Looping over atoms in reverse order

```
using System;
using OpenEye.OEChem;

public class ReverseIteration
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "n1ccccc1");

        OEAtomBaseIter aitr = mol.GetAtoms();
        for (aitr.ToLast(); aitr.IsValid(); aitr.Decrement())
        {
            Console.WriteLine(aitr.Target().GetAtomicNum());
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

2.1.5 Atom, Bond, and Conformer Indices

The following methods return the unique index assigned to its associated object upon creation.

- *OEAtomBase.GetIdx*
- *OEBondBase.GetIdx*
- *OEConfBase.GetIdx*

Note

There is a parallel method *SetIdx* as well. This is an advanced API that should never be used.

Warning

Atoms, bonds and conformers are stored inside the molecule. If the molecule is deallocated, the cached atoms, bonds, and conformers can not be accessed any longer.

Unique Identifiers

This index can be used to distinguish one object from another, as it is unique amongst all objects of the same type of the same molecule. Indices are also stable, meaning a given object will have the same index throughout its lifetime, independent of any other molecule manipulations, *e.g.*, reordering the molecule (*OEMolBase.OrderAtoms*), or the creation or deletion of other objects (*OEMolBase.NewAtom* or *OEMolBase.DeleteAtom*). The exception is the following methods.

- *OEMolBase.Sweep*
- *OEMolBase.Compress*
- *OEMolBase.UnCompress*
- *OEMCMolBase.SweepConfs*

These methods were designed with the intent of changing something about the underlying structure of the molecule, and thus are allowed to invalidate indices if they deem it necessary.

Indices can be assumed to be dense small integers greater than or equal to zero and less than the values returned by the following methods.

- *OEMolBase.GetMaxAtomIdx* for atoms
- *OEMolBase.GetMaxBondIdx* for bonds
- *OEMCMolBase.GetMaxConfIdx* for conformers

The index created on a new atom, bond, or conformer using *OEMolBase.NewAtom*, *OEMolBase.NewBond*, or *OEMCMolBase.NewConf* respectively is guaranteed to be greater than or equal to the index returned by the above methods.

Parallel Data Structures

Indices are ideal for indexing into densely packed arrays of information about the molecule. Many **OEChem TK** functions use them to this end, *e.g.*, *OEAddMols*, *OESubsetMol*, or *OEDetermineComponents*. The coordinates of a molecule are retrieved as an array of floating point values indexed by the atom indices. The following example demonstrates how to use the atom indices for a rudimentary XYZ file format writer.

Listing 1: Rudimentary XYZ writer

```
using System;
using OpenEye.OEChem;

public class XYZWriter
{
    public static int Main(string[] args)
    {
        if (args.Length != 1)
        {
            OEChem.OEThrow.Usage("XYZWriter <input>");
        }

        oemolistream ifs = new oemolistream();

        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0]);
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            Console.WriteLine(mol.NumAtoms());
            Console.WriteLine(mol.GetTitle());

            float[] coords = new float[mol.GetMaxAtomIdx() * 3];
            mol.GetCoords(coords);

            foreach (OEAtomBase atom in mol.GetAtoms())
            {
                uint idx = atom.GetIdx();

                Console.WriteLine("{0,-3}{1,11:0.00000}{2,11:0.00000}{3,11:0.00000}",
                                  OEChem.OEGetAtomicSymbol(atom.GetAtomicNum()),
                                  coords[idx * 3],
                                  coords[idx * 3 + 1],
                                  coords[idx * 3 + 2]);
            }
        }
        return 0;
    }
}
```

Computers are very efficient at doing this sort of sequential lookup. Whenever an efficient temporary data structure is needed to track information about a molecule indices should be used.

Indices for Molecule Lookup Considered Harmful

Note that atom, bond, and conformer indices are not guaranteed to be sequential, or even created sequentially, and hence atom indices can not and should not be used to retrieve all of the atoms of a molecule. Even typing the following idiom may invalidate any chance of support and eliminate any glimmer of respect from *OpenEye, Cadence Molecular Sciences* or the computational chemistry/cheminformatics community.

Warning

```
// Never ever, ever do this!!!
for (uint i=0; i<=mol.NumAtoms(); ++i)
{
   OEAtomBase atom = mol.GetAtom(new OEHasAtomIdx(i));
   // pretend atom is valid
}
```

There are far more efficient methods of crashing computer software that should be used instead.

The common misconception is that **OEChem TK** indices should be stored in order to reference back to an atom in the molecule. This leads to the idiom used in *Listing 2* which is technically legal **OEChem TK**, however, is just as insidious as the previous code snippet. The code is supposed to mimic a common technique to cache particular atoms based on some expensive to calculate property. The property in this instance is whether the atom is alpha beta unsaturated.

Warning

Listing 2: Evil atom cache

```
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class AtomSubsetEvil
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CCCC=O");

        List<uint> acache = new List<uint>(); // cache of atoms
        foreach (OEAtomBase atom in mol.GetAtoms(new OEHasAlphaBetaUnsat()))
        {
            acache.Add(atom.GetIdx()); // evil!
        }

        // pretend this code is deep in some inner loop that needs to go fast
        foreach (uint aidx in acache)
        {
            OEAtomBase catom = mol.GetAtom(new OEHasAtomIdx(aidx)); // O(n) lookup!

            // do something with the cached atom "catom"
            catom.SetName("Hello World");
```

```
        }
    }
}
```

The *OEMolBase.GetAtom* method performs a linear, $O(mol.NumAtoms())$, search over the molecule looking for the first atom that matches the predicate. This leads to a multiplicative effect when looping over multiple atoms, calling *GetAtom* for each atom. The resulting algorithm is quadratic $O(mol.NumAtoms()^2)$, possibly destroying any benefits of caching the user's expensive per atom calculation.

The proper way to store references to atoms for later use is to store the *OEAtomBase* itself. It is guaranteed that these atoms will be valid for the lifetime of the molecule.

This is regardless of molecule manipulations, *e.g.*, reordering the molecule (*OEMolBase.OrderAtoms*), or the creation or deletion of other atoms (*OEMolBase.NewAtom* or *OEMolBase.DeleteAtom*). The `Sweep` and `Compress` methods are again exceptions to this rule.

Listing 3 demonstrates the proper way to create a cache of atoms. It is important to remember that the *OEAtomBases* in the container are only valid while the molecule exists. Using the atoms after the molecule is destroyed is undefined behavior (usually a segmentation fault).

Listing 3: Atom caching

```
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class AtomSubsetGood
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CCCC=O");

        List<OEAtomBase> acache = new List<OEAtomBase>(); // cache of atoms
        foreach (OEAtomBase atom in mol.GetAtoms(new OEHasAlphaBetaUnsat()))
        {
            acache.Add(atom);
        }

        // pretend this code is deep in some inner loop that needs to go fast
        foreach (OEAtomBase catom in acache)
        {
            // do something with the cached atom "catom"
            catom.SetName("Hello World");
        }
    }
}
```

2.1.6 Atom Properties

The *OEAtomBase* class is the workhorse of the **OEChem TK** library, representing the atoms of a molecule. This section details some of its important and often used properties and methods.

Read/Write Atom Properties

Property Name	Type	Get Method	Set Method	See Also
Atomic Number	unsigned int	*GetAtomicNum*	*SetAtomicNum*	
Formal Charge	int	*GetFormalCharg*	*SetFormalCharg*	*Formal Charges*
Implicit Hyd. Count	unsigned int	*GetImplicitHCo*	*SetImplicitHCo*	
Isotopic Mass	unsigned int	*GetIsotope*	*SetIsotope*	
Partial Charge	double	*GetPartialChar*	*SetPartialChar*	*Partial Charges*
Atomic Hybridization	unsigned int	*GetHyb*	*SetHyb*	
Integer Atom Type	int	*GetIntType*	*SetIntType*	
Atom Name	string	*GetName*	*SetName*	
Atom Type Name	string	*GetType*	*SetType*	
Atom Radius	double	*GetRadius*	*SetRadius*	
Reaction Role	unsigned int	*GetRxnRole*	*SetRxnRole*	*Reactions*
Reaction Map Index	unsigned int	*GetMapIdx*	*SetMapIdx*	*Reactions*
Ring Membership	bool	*IsInRing*	*SetIsInRing*	*Ring Perception*
Aromaticity	bool	*IsAromatic*	*SetAromatic*	*Aromaticity*
Atom Chirality	bool	*IsChiral*	*SetChiral*	*Stereochemistry*
Atom Stereo	unsigned int	*GetStereo*	*SetStereo*	*Stereochemistry*

Read Only Atom Properties

Property Name	Type	Get Method	See Also
Atom Degree	unsigned int	*GetDegree*	
Explicit Atom Degree	unsigned int	*GetExplicitDegree*	
Explicit Hydrogen Count	unsigned int	*GetExplicitHCount*	
Explicit Valence	unsigned int	*GetExplicitValence*	*Valence Models*
Atom Heavy Degree	unsigned int	*GetHvyDegree*	
Atom Heavy Valence	unsigned int	*GetHvyValence*	*Valence Models*
Atom Parent	*OEMolBase*	*GetParent*	
Total Hydrogen Count	unsigned int	*GetTotalHCount*	
Atom Valence	unsigned int	*GetValence*	*Valence Models*
Atom Index	unsigned int	*GetIdx*	*Atom, Bond, and Conformer Indices*
Atom Has Stereo	bool	*HasStereoSpecifie*	*Stereochemistry Perception*
Atom Connection	bool	*IsConnected*	

Attach Generic Data to Atoms

Generic data (see *Generic Data*) can be attached to any object that derives from the *OEBase* class. The following program shows an example where hydrogen bonding donor property is attached as a `bool` value to the corresponding *OEAtomBase* object.

Listing 2: Example of attaching generic data to atoms

```
using System;

using OpenEye.OEChem;

public class GenericDataToAtom
```

(continues on next page)

(continued from previous page)

```
{
    private class IsDonorAtomPred : OEUnaryAtomPred
    {
        public override bool Call(OEAtomBase atom)
        {
            return atom.GetBoolData("isdonor");
        }
        public override OEUnaryAtomBoolFunc CreateCopy()
        {
            OEUnaryAtomBoolFunc copy = new IsDonorAtomPred();
            copy.ReleaseOwnership();
            return copy;
        }
    }

    public static void Main(String[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1c(Cl)cncc1C(=O)O");

        OEMatchAtom IsDonorAtom = new OEMatchAtom("[!H0;#7,#8]");
        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            atom.SetBoolData("isdonor", IsDonorAtom.Call(atom));
        }

        Console.Write("Donor atoms: ");
        foreach (OEAtomBase atom in mol.GetAtoms(new IsDonorAtomPred()))
        {
            Console.Write(atom.GetIdx() + " " +  OEChem.OEGetAtomicSymbol(atom.
↪GetAtomicNum()));
        }
        Console.WriteLine("");
    }
}
```

The *OEAtomBase* API provides the following methods, publicly inherited from *OEBase*, that allow the manipulation of generic data.

Table 9: **Methods to manipulate generic data**

Method	Description
SetData	sets a generic data associating it with the given tag
AddData	adds a generic data associating it with the given tag
HasData	determines whether a molecule has any generic data with a given tag
GetData	returns the generic data associated with the given tag
DeleteData	deletes all generic data with the given tag
Clear	clears all stored generic data

Note

Generic data attached to a molecule or any of its atoms or bonds is automatically saved when the molecule is written into an `.oeb` file.

See also

- *Attaching other objects* section
- *SD Tagged Data Manipulation* section
- *PDB Tagged Data Manipulation* section

2.1.7 Bond Properties

The `OEBondBase` represents the bonds of a molecule. This section details some of its important and often used properties and methods.

Read/Write Bond Properties

Property Name	Type	Get Method	Set Method	See Also
Bond Order	unsigned int	`GetOrder`	`SetOrder`	
Aromaticity	bool	`IsAromatic`	`SetAromatic`	*Aromaticity Perception*
Ring Membership	bool	`IsInRing`	`SetInRing`	*Ring Perception*
Integer Bond Type	int	`GetIntType`	`SetIntType`	
Bond Type Name	string	`GetType`	`SetType`	
Begin Atom	`OEAtomBase`	`GetBgn`	`SetBgn`	
End Atom	`OEAtomBase`	`GetEnd`	`SetEnd`	
Bond Chirality	bool	`IsChiral`	`SetChiral`	*Stereochemistry Perception*
Bond Stereo	unsigned int	`GetStereo`	`SetStereo`	*Stereochemistry Perception*

Read Only Bond Properties

Property Name	Type	Get Method	See Also
Begin Atom Index	unsigned int	`GetBgnIdx`	*Atom, Bond, and Conformer Indices*
End Atom Index	unsigned int	`GetEndIdx`	*Atom, Bond, and Conformer Indices*
Bond Index	unsigned int	`GetIdx`	*Atom, Bond, and Conformer Indices*
Bond Has Stereo	bool	`HasStereoSpecifiec`	*Stereochemistry Perception*
Bond Parent	`OEMolBase`	`GetParent`	
Rotatable	bool	`IsRotor`	rotatable bond

Attach Generic Data to Bonds

Generic data (see *Generic Data*) can be attached to any object that derives from the *OEBase* class. The *OEBondBase* API provides the following methods, publicly inherited from *OEBase*, that allow the manipulation of generic data.

Table 10: **Methods to manipulate generic data**

Method	Description
`SetData`	sets a generic data associating it with the given tag
`AddData`	adds a generic data associating it with the given tag
`HasData`	determines whether a molecule has any generic data with a given tag
`GetData`	returns the generic data associated with the given tag
`DeleteData`	deletes all generic data with the given tag
`Clear`	clears all stored generic data

Note

Generic data attached to a molecule or any of its atoms or bonds is automatically saved when the molecule is written into an `.oeb` file.

See also

- *Attaching other objects* section
- *SD Tagged Data Manipulation* section
- *PDB Tagged Data Manipulation* section

2.1.8 Valence Models

This section describes the three valence models currently implemented by **OEChem TK**.

1. For molecules that have fully specified formal charges, the MDL valence model may be used to assign hydrogen counts. (See section *MDL Valence Model*.)
2. For molecules that have fully specified hydrogen counts, the OpenEye ‘’charge” model may be used to assign formal charges. (See section *OpenEye Charge Model*.)
3. Finally, for molecules with neither formal charges nor hydrogen counts, **OEChem TK** uses the OpenEye hydrogen count model to assign both hydrogen counts and formal charges. (See section *OpenEye Hydrogen Count Model*.)

MDL Valence Model

See also

The following APIs support the **MDL valence model**:

- `OEDefaultMDLHCount` function
- `OEAssignMDLHydrogens` function
- `OEMDLGetValence` function
- `OEIsValidMDLAtomValence` predicate

The MDL valence model was developed by MDL to allow hydrogen counts to be implicit in MDL SD and MOL file formats. MDL valence is simply the number of covalent bonds to an atom (i.e., the sum of the bond orders), not the

number of bonded neighbors. It assumes that the bond orders to an atom are specified (explicit valence), and that the atomic number and formal charge are correctly set. The MDL valence model then prescribes the number of implicit hydrogens on a particular atom. The periodic table graphic *below* shows the MDL valence model as implemented in **OEChem TK**.

The valence rules are,

- For transition metals, lanthanides and actinides, all valences are allowed which means that implicit hydrogen counts will never be applied.
- For the main group elements, allowed valences are listed parenthetically for each atom in the graphic below.
- Charges are handled by a simple shift: for positive and negative charges, convert the charged atom to its isoelectronic neutral atom; positive charges move atoms to the left and negative charges move the atom to the right in the periodic table, skipping the transition metal block. Thus, `C+` is considered to be isoelectronic to `B`, `C(+2)` is equivalent to `Be`, `C(-1)` is equivalent to `N`, and `C(-2)` is equivalent to `O`.
- If the isolectronic version of a charged atom is a Group I or II atom, no implicit hydrogen counts are applied (e.g., `Sn(+2)`)
- A charged atom is considered an illegal valence if its neutral isolectronic atom type wraps beyond a noble gas configuration (e.g., `Li(+2)` or `F(-2)`)

1 H (1)																	2 He (0)
3 Li (1)	4 Be (2)											5 B (3)	6 C (4)	7 N (3,5)	8 O (2)	9 F (1)	10 Ne (0)
11 Na (1)	12 Mg (2)											13 Al (3)	14 Si (4)	15 P (3,5)	16 S (2,4,6)	17 Cl (1,3,5,7)	18 Ar (0)
19 K (1)	20 Ca (2)	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga (3)	32 Ge (4)	33 As (3,5)	34 Se (2,4,6)	35 Br (1,3,5,7)	36 Kr (0)
37 Rb (1)	38 Sr (2)	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In (3)	50 Sn (2,4)	51 Sb (3,5)	52 Te (2,4,6)	53 I (1,3,5,7)	54 Xe (0)
55 Cs (1)	56 Ba (2)		72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl (1,3)	82 Pb (2,4)	83 Bi (3,5)	84 Po (2,4,6)	85 At (1,3,5,7)	86 Rn (0)
87 Fr (1)	88 Ra (2)		104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg							

57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
89 Ac	90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

Fig. 1: **MDL Valence Model**

OpenEye Valence Model

See also

The following APIs support the **OpenEye valence model**:

- *OEDefaultImplicitHCount* function
- *OEAssignImplicitHydrogens* function
- *OEAssignFormalCharges* function
- *OECheckAtomValence* function
- *OECheckAtomValences* function
- *OEIsValidAtomValence* predicate

OpenEye Charge Model

The *OpenEye* formal charge model assigns formal charges to elements based upon their total valence. In **OEChem TK**, this functionality is invoked by the *OEAssignFormalCharges* function. If the formal charge on an atom is non-zero, it is left unchanged.

Hydrogen
If the valence isn't one, the formal charge is +1.

Boron
If the valence is four, the formal charge is +1.

Carbon
If the valence is three, the formal charge is +1 if the atom has a polar neighbor, *i.e.* N, O or S, and formal charge -1 otherwise.

Nitrogen
If the valence is two, the formal charge is -1, and if the valence is four the formal charge is +1.

Oxygen
If the valence is one, the formal charge is -1, and if the valence is three the formal charge is +1.

Phosphorous
If the valence is four, the formal charge is +1.

Sulfur
If the valence is 1, the formal charge is -1, if the valence is three the formal charge is +1, if the valence is 5, the formal charge is -1, if the valence is four and the degree is four the charge is +2.

Chlorine
If the valence is 0 the formal charge is -1, if the valence is four the formal charge is +3.

Fluorine, Bromine, Iodine
If the valence is zero, the formal charge is -1.

Magnesium, Calcium, Zinc
If the valence is zero, the formal charge is +2.

Lithium, Sodium, Potassium
If the valence is zero, the formal charge is +1.

Iron
If the valence is zero, the formal charge is +3 if the partial charge is 3.0, and +2 otherwise.

Copper
If the valence is zero, the formal charge is +2 if the partial charge is 2.0, and +1 otherwise.

For the remaining elements, if the valence of an atom is zero, its formal charge is set from its partial charge.

OpenEye Hydrogen Count Model

OpenEye's hydrogen count valence model is used by **OEChem TK** when neither hydrogen counts nor valence are specified. The typical uses are reading molecules from PDB or XYZ format files without explicit hydrogens. This functionality is invoked by *OEAssignImplicitHydrogens*, which must always be followed by a call to *OEAssignFormalCharges*. This valence model is unique in that it only partially updates hydrogen counts, assuming the unfilled valences will be corrected by *OpenEye*'s charge valence model above. In MDL's model for example, a neutral sodium atom is assumed to have one implicit hydrogen, *i.e.* sodium hydride instead of sodium metal. In *OpenEye*'s hydrogen count valence model, a disconnected sodium atom is assumed to be a sodium cation, `[Na+]`. When reading from PDB files, this is a very reasonable assumption.

Note that although the *OpenEye* hydrogen count valence model often sets charge and protonation states to physiological conditions, it is neither intended to be a pKa nor ionization state predictor. Instead, it is a normalization. Much like many registry systems and the MDL valence model which will convert `C(=O)[O-]` to `C(=O)O` for registration purposes, this valence model converts the opposite direction to `C(=O)[O-]`.

- **Carbon** is always assumed to be at least four valent, and therefore neutral.
- **Nitrogens** that are conjugated (have double bonds, or have neighbors that have double bonds, in their Kekulé representations) are assumed at least three valent and neutral, whilst all other nitrogens are assumed to be (minimum) four valent, with a +1 formal charge.
- **Oxygens** are assumed to be at least two valent and neutral, unless they have a single bond to an atom that is doubly bonded to oxygen, in which case it's assumed to be one valent, with a -1 formal charge.
- **Sulfur** is assumed to always be at least two valent.

All other elements are assumed to have no implicit hydrogens, and the formal charge as specified by the *OpenEye* charge model. This models all disconnected halogens as halide anions, and when disconnected the metals listed above as cations.

These rules are sufficient to reasonably protonate proteins read from PDB files. However, as described above, these rules are not intended to be a comprehensive rule-based pKa predictor. Users interested in predicting physiological ionization, and protonation/disassociation state enumeration should contact *OpenEye, Cadence Molecular Sciences* (http://www.eyesopen.com/) about our tools for exactly this task.

Molecules

2.1.9 Molecule Properties

Molecules in **OEChem TK** are represented by *OEMolBases*. In addition to keeping track of the atoms and bonds that constitute a molecule, the *OEMolBase* is also used to store global information and *Generic Data* about the molecule.

Read/Write Molecule Properties

The *OEMolBase* API provides the following methods for storing common global information about the molecule.

Property Name	Type	Get Method	Set Method	See Also
Coordinates	`float` array	*GetCoords*	*SetCoords*	*ClearCoords*
Dimension	`unsigned int`	*GetDimension*	*SetDimension*	
Energy	`float`	*GetEnergy*	*SetEnergy*	
Perception	`bool`	*HasPerceived*	*SetPerceived*	
Rxn	`bool`	*IsRxn*	*SetRxn*	*Reactions*
Title	`string`	*GetTitle*	*SetTitle*	*Title*

Title

The **Title** property is a string used to represent the name of the molecule. The default value is an empty string. This field may be used to store a registry number or other identifier, instead of a common name. The string is typically trimmed of white space by most file format readers.

The following code uses the *GetTitle* method to list the names of the molecules in a file. The input file is read and the list of identifiers (molecule names) are written to standard-out.

Listing 1: Printing molecule titles

```
using System;
using OpenEye.OEChem;

public class GetTitle
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        oemolistream ifs = new oemolistream(args[0]);

        while (OEChem.OEReadMolecule(ifs, mol))
        {
            Console.WriteLine(mol.GetTitle());
        }
        return 0;
    }
}
```

Atoms and Bonds

The *OEMolBase* API provides the following methods for accessing and altering the connection table of the molecule.

Description	Return Type	Method	See Also
Number of Atoms	unsigned int	*NumAtoms*	
Number of Bonds	unsigned int	*NumBonds*	
Maximum Atom Index	unsigned int	*GetMaxAtomIdx*	*Atom, Bond, and Conformer Indices*
Maximum Bond Index	unsigned int	*GetMaxBondIdx*	*Atom, Bond, and Conformer Indices*
Access to an Atom	*OEAtomBase*	*GetAtom*	*Predicate Functors*
Access to a Bond	*OEBondBase*	*GetBond*	*Predicate Functors*
Access to all Atoms	*OEIterBase<OE*	*GetAtoms*	*Atom and Bond Traversal*
Access to all Bonds	*OEIterBase<OE*	*GetBonds*	*Atom and Bond Traversal*
Create a new Atom	*OEAtomBase*	*NewAtom*	*Atom and Bond Creation*
Create a new Bond	*OEBondBase*	*NewBond*	*Atom and Bond Creation*
Remove an Atom	bool	*DeleteAtom*	
Remove a Bond	bool	*DeleteBond*	
Rearrange Atoms	bool	*OrderAtoms*	
Rearrange Bonds	bool	*OrderBonds*	

Atom and Bond Creation

OEChem TK contains functions that allow molecules to be constructed from atoms and bonds explicitly. *Listing 2* demonstrates how to create the water molecule.

Atoms are created by calling the *OEMolBase.NewAtom* method, and bonds are created by calling the *OEMolBase.NewBond* method. *NewAtom* takes the atomic number of the atom to create and returns the new *OEAtomBase*. *NewBond* takes two *OEAtomBases* and an integer bond order as arguments and returns the new *OEBondBase*.

Listing 2: Creating new atoms and bonds

```
using System;
using OpenEye.OEChem;

public class CreatingAtoms
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();

        OEAtomBase o  = mol.NewAtom(8);
        OEAtomBase h1 = mol.NewAtom(1);
        OEAtomBase h2 = mol.NewAtom(1);

        OEBondBase b1 = mol.NewBond(o, h1, 1);
        OEBondBase b2 = mol.NewBond(o, h2, 1);
    }
}
```

In *Listing 2* the atomic numbers of oxygen, 8, and hydrogen, 1, are explicitly encoded in the program. To make code easier to read and less error prone, **OEChem TK** provides symbolic constants for the first 109 elements in the *OEElemNo* namespace. The *OEElemNo* namespace defines the atomic symbols with the appropriate atomic number.

Listing 3: Using OEElemNo namespace

```
using System;
using OpenEye.OEChem;

public class CreatingAtoms2
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();

        OEAtomBase o  = mol.NewAtom(OEElemNo.O);
        OEAtomBase h1 = mol.NewAtom(OEElemNo.H);
        OEAtomBase h2 = mol.NewAtom(OEElemNo.H);

        OEBondBase b1 = mol.NewBond(o, h1, 1);
        OEBondBase b2 = mol.NewBond(o, h2, 1);
    }
}
```

Note

The atoms and bonds of a molecule are automatically deleted when their parent molecule is destroyed.

Attach Generic Data to Molecules

Generic data can be attached to any object that derives from the *OEBase* class. The following two snippets demonstrate how generic data (e.g. molecule weight) can be attached to a molecule:

```
uint tag = OEChem.OEGetTag("MolWeight");
mol.SetDoubleData(tag, OEChem.OECalculateMolecularWeight(mol));
```

```
mol.SetDoubleData("MolWeight", OEChem.OECalculateMolecularWeight(mol));
```

After annotation, the data can be accessed with the same integer or character string identifier:

```
tag = OEChem.OEGetTag("MolWeight");
Console.WriteLine(mol.GetDoubleData(tag));
```

```
Console.WriteLine(mol.GetDoubleData("MolWeight"));
```

The *OEMolBase* API provides the following methods, publicly inherited from *OEBase*, that allow the manipulation of generic data.

Table 11: **Methods to manipulate generic data**

Method	Description
SetData	sets a generic data associating it with the given tag
AddData	adds a generic data associating it with the given tag
HasData	determines whether a molecule has any generic data with a given tag
GetData	returns the generic data associated with the given tag
DeleteData	deletes all generic data with the given tag
Clear	clears all stored generic data

Note

Generic data attached to a molecule or any of its atoms or bonds is automatically saved when the molecule is written into an `.oeb` file.

See also

- *Attaching other objects* section
- *SD Tagged Data Manipulation* section
- *PDB Tagged Data Manipulation* section
- For using tag data with multi-conformer molecules, see *Dude, where's my SD data?*

2.1.10 Ring Perception

Cycle Membership

The simplest form of ring processing in **OEChem TK** is testing whether an atom or bond is in a ring or not. The **OEChem TK** function *OEFindRingAtomsAndBonds* is used to determine which atoms and bonds are members of one or more rings and which are acyclic. This function uses an efficient $O(n)$ algorithm. Once *OEFindRingAtomsAndBonds* has been called, an atom or bond can be tested for being in a ring by calling the *OEAtomBase.IsInRing* or the *OEBondBase.IsInRing* methods respectively.

The function *OEFindRingAtomsAndBonds* is called automatically by the high-level file read function *OEReadMolecule* and *OESmilesToMol*. However, whenever you modify a molecule by adding or deleting bonds, you'll need to explicitly call *OEFindRingAtomsAndBonds*.

The following two 'equivalent' code snippets demonstrate how to loop over chain atoms using the *OEAtomIsInRing* functor and the *IsInRing* method of the *OEAtomBase* class.

```
foreach (OEAtomBase atom in mol.GetAtoms(new OENotAtom(new OEAtomIsInRing())))
{
    Console.WriteLine(atom.GetIdx() + " " + OEChem.OEGetAtomicSymbol(atom.
↪GetAtomicNum()));
}
```

```
foreach (OEAtomBase atom in mol.GetAtoms())
{
    if (!atom.IsInRing())
    {
        Console.WriteLine(atom.GetIdx() + " " + OEChem.OEGetAtomicSymbol(atom.
↪GetAtomicNum()));
    }
}
```

The chain/ring bonds of a molecule can similarly be accessed using the *OEBondIsInRing* functor and the *IsInRing* method of the *OEBondBase* class. For more information about functors see chapter *Predicate Functors*.

The user can also set the atom and bond ring flags manually using the *OEAtomBase.SetInRing* and *OEBondBase.SetInRing* methods.

Code Example

- Ring Perception **OpenEye Python Cookbook** recipe

Membership in a Given Ring Size

It is also possible to use **OEChem TK** to determine whether an atom or a bond is in a ring of a given size, using the *OEAtomIsInRingSize* and *OEBondIsInRingSize* functions.

Both of these functions require that *OEFindRingAtomsAndBonds* has previously been called on the molecule. Both of these functions take the query ring size as an argument, which should be greater than or equal to three. The definition of ring or cycle is not based upon the ''smallest set of smallest rings" (SSSR), and the functions return true if there is a bonded path of *size* unique atoms where each atom is bonded to the next and the last is bonded to the first.

It is often the case that atoms may be in different sized cycles at the same time. For example, one way to identify the ring fusion atoms in indole (the fusion of a five-membered pyrrole ring and a six-membered benzene ring) is the following:

```
foreach (OEAtomBase atom in mol.GetAtoms())
{
    if (OEChem.OEAtomIsInRingSize(atom, 5) &&
            OEChem.OEAtomIsInRingSize(atom, 6))
    {
        Console.WriteLine(atom.GetIdx());
    }
}
```

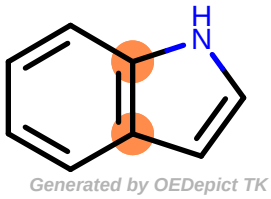


Fig. 2: **The fused atoms in indole belong to both a five- and a six-membered rings**

OEChem TK also provides an additional pair of functions, *OEAtomIsInAromaticRingSize* and *OEBondIsInAromaticRingSize*, to determine whether an atom or bond is in an aromatic ring or cycle of a given size. These behave identically to *OEAtomIsInRingSize* and *OEBondIsInRingSize* except that each ring bond in the path/cycle must be aromatic. In addition to *OEFindRingAtomsAndBonds*, these functions also require the user to have called *OEAssignAromaticFlags*.

Code Example

- Ring Perception **OpenEye Python Cookbook** recipe

Smallest Ring Membership

In addition to determining whether an atom or a bond is in a ring or cycle of a given size, it's often useful to know the size of the smallest ring or cycle that an atom or bond is in. To do this **OEChem TK** provides the functions *OEAtomGetSmallestRingSize* and *OEBondGetSmallestRingSize*. For acyclic atoms and bonds, these functions return the value zero. For cyclic atoms and bonds, they return a value greater than or equal to three.

```
uint size;
foreach (OEAtomBase atom in mol.GetAtoms())
{
    size = OEChem.OEAtomGetSmallestRingSize(atom);
    if (size == 0)
    {
        Console.WriteLine(atom.GetIdx() + " acyclic");
    }
    else
    {
        Console.WriteLine(atom.GetIdx() + " smallest ring size= " + size);
    }
}
```

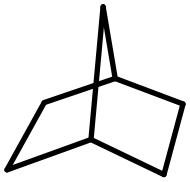


Fig. 3: **OEAtomGetSmallestRingSize returns five for each heavy atom of norbornane**

Code Example

- Ring Perception **OpenEye Python Cookbook** recipe

Connected Components Identification

To aid in splitting molecules into discrete connected components, for example to separate a parent compound from its salt, or a ligand from a protein, **OEChem TK** provides the function *OEDetermineComponents*. This function arbitrarily assigns an integer index, starting from one, to each disconnected part in the *OEMolBase*.

On return this provides a mapping from each atom's index, obtained by *OEAtomBase.GetIdx*, to its component index. Unused atom indices are mapped to zero. The function itself also returns the total number of components found, *i.e.* the maximum part index stored in the array. The following snippet provides a short example of how to use this function.

```
public static void MoleculeParts(OEMolBase mol)
{
    uint[] parts = new uint[mol.GetMaxAtomIdx()];

    uint count = OEChem.OEDetermineComponents(mol, parts);
    Console.WriteLine("The molecule has " + count + " components");

    foreach (OEAtomBase atom in mol.GetAtoms())
    {
        Console.WriteLine("atom " + atom.GetIdx() + " is in part " + parts[atom.
↪GetIdx()]);
    }
}
```

Code Example

- *Splitting multicomponent molecules* **OEChem TK** example
- Ring Perception **OpenEye Python Cookbook** recipe

Ring Systems Identification

The **OEChem TK** function *OEDetermineRingSystems* and *OEDetermineAromaticRingSystems* behave very similarly to the *OEDetermineComponents*. However, these functions return a mapping from atom indices to a ring or aromatic ring system index, respectively (see example in *Listing 1*). These functions require that *OEFindRingAtomsAndBonds* has been called previously. The function *OEDetermineAromaticRingSystems* also requires the aromaticity perception by calling the *OEAssignAromaticFlags* function. The *OESmilesToMol* used in

the above example automatically calls both the *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags* functions.

When using the *OEDetermineRingSystems* function, all acyclic atoms are mapped to the value zero. When using the *OEDetermineAromaticRingSystems* function, all aliphatic atoms are mapped to the value zero.

Listing 1: Aromatic ring system identification

```
using System;
using OpenEye.OEChem;

public class PerceiveAromRingSystems
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "C(O)(=O)c1cccc2c1[nH]c(C3CCCc4c3cccc4)c2");

        uint[] parts = new uint[mol.GetMaxAtomIdx()];

        uint nraromsystems = OEChem.OEDetermineAromaticRingSystems(mol, parts);

        Console.Write("Aliphatic atoms: ");
        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            if (parts[atom.GetIdx()] == 0)
            {
                Console.Write(atom.GetIdx().ToString() + " ");
            }
        }

        Console.WriteLine();
        Console.WriteLine("Number of aromatic ring systems = " + nraromsystems.
↪ToString());
        for (uint ringidx = 1; ringidx <= nraromsystems; ++ringidx)
        {
            Console.Write(ringidx.ToString() + " . aromatic ring system: ");

            foreach (OEAtomBase atom in mol.GetAtoms())
            {
                if (parts[atom.GetIdx()] == ringidx)
                {
                    Console.Write(atom.GetIdx().ToString() + " ");
                }
            }

            Console.WriteLine();
        }
        return 0;
    }
}
```

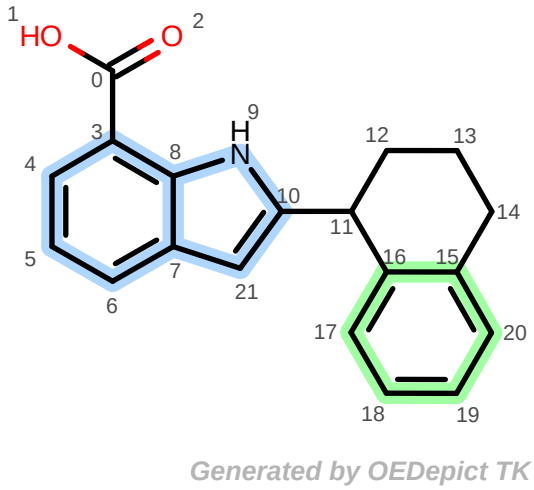


Fig. 4: **Example structure of aromatic ring system identification**

The output of *Listing 1* is the following:

```
Aliphatic atoms: 0 1 2 11 12 13 14
Number of aromatic ring systems = 2
1 . aromatic ring system: 3 4 5 6 7 8 9 10 21
2 . aromatic ring system: 15 16 17 18 19 20
```

Code Example

- Ring Perception **OpenEye Python Cookbook** recipe

Smallest Set of Smallest Rings (SSSR) Considered Harmful

In graph-theoretical terms, a bond is considered cyclic if its removal from the structure would not lead to the structure being broken into separate components. Despite this simple definition, a large number of algorithms for ring detection exist. (See article [Downs-1989] for an extensive and comparative review.) The diversity and multitude of the ring perception methods derives from the fact that, while determining whether an atom is part of a ring is a very simple problem, the identification of "chemically meaningful" rings among the potentially large number of cyclic subgraphs of a molecular structure can be a surprisingly complex task.

The *Smallest set of smallest rings* (SSSR) [Plotkin-1971] is the most broadly used type of ring set in computational chemistry. However SSSR is not a unique subset of all possible cycles of a molecule (see *Figure: Example of SSSR*). Obviously SSSR membership can not be used as a graph theoretical invariant in symmetry perception. Indeed the choice of which rings are part of the SSSR and which are not is arbitrary, and often dependent upon the input order of the molecule. Because of the potential ambiguity of SSSR, many alternative ring set definitions to SSSR have been proposed over the years, including extended SSSR, the set of "synthetically important" rings, the set of elementary rings (SER), the essential set of essential rings (ESER), κ-rings, etc…

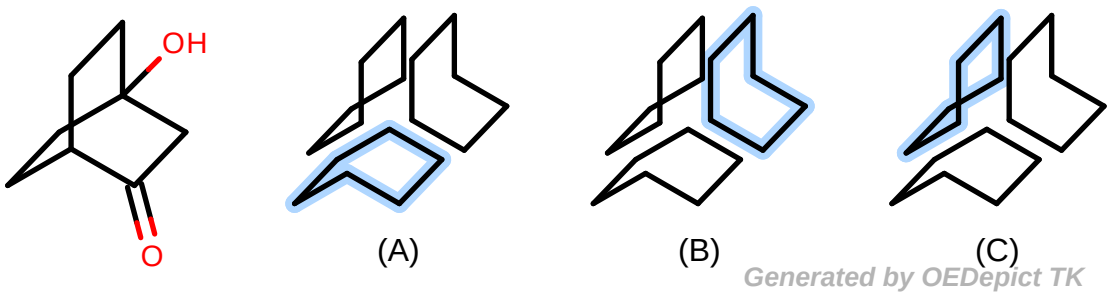


Fig. 5: **Example of SSSR** *SSSR is not an invariant subset of all possible rings; (a), (b) and (c) depict the three equally valid SSSR of the bridged structure (on the left)*

We believe that it is a great service to our customers that we do not include any SSSR functionality in **OEChem TK**. This is a conscious (and consensus) decision. The forerunners of **OEChem TK**, Babel and OELib, both contained efficient algorithms for determining SSSR, and these remain freely available on the Internet today. Furthermore, many useful ring perception routines are available in **OEChem TK**, including:

- the ability to detect whether an atom or bond is acyclic or part of a ring (For more details on `OEAtomBase.IsInRing` and `OEBondBase.IsInRing`, see section *Cycle Membership*)
- the ability to detect whether an atom or a bond is in a ring of a particular size (For more details on `OEAtomIsInRingSize` and `OEBondIsInRingSize`, see section *Membership in a Given Ring Size*)
- the ability to detect whether an atom or a bond is in an aromatic ring of a particular size (For more details on `OEAtomIsInAromaticRingSize` and `OEBondIsInAromaticRingSize`, see section *Membership in a Given Ring Size*)
- the size of the smallest ring an atom or bond belongs to (For more details on `OEAtomGetSmallestRingSize` and `OEBondGetSmallestRingSize`, see section *Smallest Ring Membership*)
- the ability to determine whether atoms belong to the same ring system (`OEDetermineRingSystems`) or the same aromatic ring system (`OEDetermineAromaticRingSystems`). See section *Ring Systems Identification* for more details.

See also

- [Balducci-1994]
- [Baumer-1991]
- [Downs-1989]
- [Figueras-1996]
- [Fujita-1988]
- [Gasteiger-1979]
- [Hanser-1996]
- [Plotkin-1971]
- [Qian-1990]
- [Roos-Kozel-1981]
- [Takahashi-1994]

- [Zamora-1976]

2.1.11 Aromaticity Perception

Aromaticity and Hückel's rule

OEChem TK's aromaticity perception routines are based around the Hückel's rule that defines cyclic conjugated systems with $(4N + 2)$ number of π electrons as aromatic (where N is zero or any positive integer).

Aromaticity can be set using the *OEAssignAromaticFlags* function, which takes an *OEMolBase* argument. The *OEAssignAromaticFlags* sets the aromaticity flags on atoms and bonds using an aromaticity model. (For the list of available aromaticity models in *OEChem TK** see section *Aromaticity Models in OEChem TK*.)

```
OEGraphMol mol = new OEGraphMol();
OEChem.OEParseSmiles(mol, "C1[NH]C=CC=1CO");
OEChem.OEAssignAromaticFlags(mol);
```

Hint

The *OESmilesToMol* function automatically perceives the aromaticity of the molecule using the *default* *OEAroModel.OpenEye* aromaticity model.

The following two code snippets demonstrate how to loop over aromatic atoms using the *OEIsAromaticAtom* functor and the *IsAromatic* method of the *OEAtomBase* class.

```
foreach (OEAtomBase atom in mol.GetAtoms(new OEIsAromaticAtom()))
{
    Console.WriteLine(atom.GetIdx() + " " +
                      OEChem.OEGetAtomicSymbol(atom.GetAtomicNum()));
}
```

```
foreach (OEAtomBase atom in mol.GetAtoms())
{
    if (atom.IsAromatic())
    {
        Console.WriteLine(atom.GetIdx() + " " +
                          OEChem.OEGetAtomicSymbol(atom.GetAtomicNum()));
    }
}
```

The aromatic bonds of a molecule can similarly be accessed using the *OEIsAromaticBond* functor and the *IsAromatic* method of the *OEBondBase* class. For more information about functors see chapter *Predicate Functors*.

The user can also set the atom and bond aromaticity flags manually using the *OEAtomBase.SetAromatic* and *OEBondBase.SetAromatic* methods.

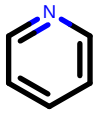
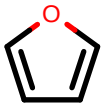
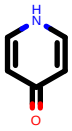
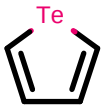
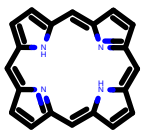
Aromaticity Models in OEChem TK

The *OEAssignAromaticFlags* function can also take an aromaticity model constant as an argument to perceive various aromaticity models. The following aromaticity models are available in **OEChem TK**:

1. *OpenEye* (default model)
2. *Daylight*

3. *Tripos*
4. *MDL*
5. *MMFF*

The following table demonstrates the differences between the five available aromaticity models.

	MDL	Tripos	MMFF	DayLight	OpenEye
	yes	yes	yes	yes	yes
	no	no	yes	yes	yes
	no	no	no	yes	yes
	no	no	no	yes	yes
	no	N/A[1]	N/A[1]	no	yes
	no	no	no	yes/no[2]	yes

Generated by OEDepict TK

Table footnotes:

[1] *Atomic elements such as Te, B, Se are not available in the MMFF and Tripos aromaticity models.*

[2] *Only two out of the four five-membered rings are recognized as aromatic.*

The code in *Listing 1* demonstrates how to perceive aromaticity with these available models.

Listing 1: Aromaticity perception with various models

```
using System;
using OpenEye.OEChem;

public class Aromaticity
{
    public static void PerceiveAromaticity(OEMolBase mol, string modelname,
                                           uint model)
    {
        OEChem.OEAssignAromaticFlags(mol, model);
        string cansmi = OEChem.OECreateCanSmiString(mol);
        Console.WriteLine(modelname + " : " + cansmi);
    }

    public static void Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ncncc1c2cocc2-c3[nH]ccc(=O)c3");

        PerceiveAromaticity(mol, "OEAroModelOpenEye ", OEAroModel.OpenEye);
        PerceiveAromaticity(mol, "OEAroModelDaylight", OEAroModel.Daylight);
        PerceiveAromaticity(mol, "OEAroModelTripos  ", OEAroModel.Tripos);
        PerceiveAromaticity(mol, "OEAroModelMMFF    ", OEAroModel.MMFF);
        PerceiveAromaticity(mol, "OEAroModelMDL     ", OEAroModel.MDL);
    }
}
```

Since these models define aromaticity rules differently, the generated canonical SMILES depend on the applied aromaticity models. The output of *Listing 1* is the following:

```
OEAroModel::OpenEye  : c1c[nH]c(cc1=O)c2cocc2c3cncnc3
OEAroModel::Daylight : c1c[nH]c(cc1=O)c2cocc2c3cncnc3
OEAroModel::Tripos   : c1c(cncn1)C2=COC=C2C3=CC(=O)C=CN3
OEAroModel::MMFF     : c1c(cncn1)c2cocc2C3=CC(=O)C=CN3
OEAroModel::MDL      : c1c(cncn1)C2=COC=C2C3=CC(=O)C=CN3
```

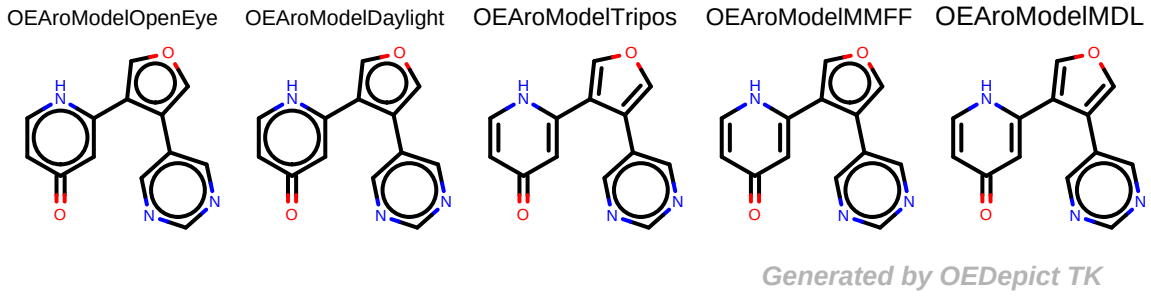


Fig. 6: *Example of aromaticity perception with different models*

Clearing Aromaticity

The aromatic property of all atoms and bonds in a molecule, can conveniently be cleared (*i.e.* set to value false) by calling the *OEClearAromaticFlags* function. This is useful when writing the Kekulé form of a SMILES string, which can be done by calling *OEClearAromaticFlags* before calling *OEKekulize* and *OECreateSmiString* functions.

Listing 2: Clearing aromaticity

```
using System;
using OpenEye.OEChem;

public class ClearAromaticity
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OEParseSmiles(mol, "n1ccncc1");
        string smiles = OEChem.OECreateCanSmiString(mol);
        Console.WriteLine("Canonical smiles : {0}", smiles);

        OEChem.OEClearAromaticFlags(mol);
        OEChem.OEKekulize(mol);
        smiles = OEChem.OECreateCanSmiString(mol);
        Console.WriteLine("Kekule smiles    : {0}", smiles);
    }
}
```

The output of *Listing 2* is the following:

```
Canonical smiles : c1cnccn1
Kekule smiles    : C1=CN=CC=N1
```

Input/Output Aromaticity

Since *OEParseSmiles* preserves the aromaticity present (or absent) in the input SMILES string, the *OEClearAromaticFlags* or the *OEAssignAromaticFlags* have to be explicitly called to remove or perceive aromaticity in a molecule, respectively. (See examples in *Listing 1* and *Listing 2*)

However, when a molecule is imported from a file with a high-level *OEReadMolecule* function, atom and bond aromaticity is automatically perceived using the default *OEAroModelOpenEye* model.

As mentioned before (in section *Molecular Property Preservation*), the high-level *OEWriteMolecule`* writer function may automatically update atom and bond properties (including aromaticity) in order to standardize the exported molecules. The following table shows the aromaticity models associated with various file formats.

File Format	Default Output Aromaticity Models
OEFormat.CAN	*OEAroModel.OpenEye*
OEFormat.CDX	*OEAroModel.OpenEye*
OEFormat.CSV	*OEAroModel.OpenEye*
OEFormat.FASTA	[1]
OEFormat.ISM	*OEAroModel.OpenEye*
OEFormat.MDL	*OEClearAromaticFlags*
OEFormat.MF	[1]
OEFormat.MMOD	[1]
OEFormat.MOL2	*OEAroModel*
OEFormat.MOL2H	*OEAroModel.Tripos*
OEFormat.MOPAC	[1]
OEFormat.OEB	[1]
OEFormat.PDB	[1]
OEFormat.SDF	*OEClearAromaticFlags*
OEFormat.SLN	*OEAroModel.Tripos*
OEFormat.SMI	*OEAroModel.OpenEye*
OEFormat.XYZ	[1]
OEFormat.INCHI	[1]
OEFormat.INCHIKEY	[1]

Table footnote:

[1] *The aromaticity model is not changed by the associated molecule writer.*

The following snippet shows how to overwrite the default aromaticity model of a specific file format.

```
oemolostream ofs = new oemolostream(".smi");
OEChem.OEWriteMolecule(ofs, mol);  // using default OpenEye aromaticity model
uint flavor = ofs.GetFlavor(ofs.GetFormat());
flavor |= OEOFlavor.Generic.OEAroModelMDL;
ofs.SetFlavor(ofs.GetFormat(), flavor);
OEChem.OEWriteMolecule(ofs, mol);
```

This will produce the following output:

```
c1c[nH]cc1CO
C1=CNC=C1CO
```

See also

- [Lloyd-1996]

2.1.12 Connectivity Perception

OEChem TK provides several functions for determining the connectivity and/or bond orders from various input file formats. For correct molecule processing, **OEChem TK** requires all the covalent bonds to be represented in a molecule and each bond to have a defined bond order, 1 for single, 2 for double, 3 for triple and 4 for quadruple. Given this explicit Kekulé representation of a molecule, **OEChem TK** can perceive and re-perceive high order attributes such as ring membership or aromaticity as defined by different aromaticity models.

Alas, unlike MDL's SD file format, not all file formats explicitly specify a Kekulé form of a molecule with explicit bond orders. The functions, described in this chapter, attempt to deduce such a representation from the information

that is available in such file formats.

Connectivity From 3D Coordinates

For file formats that provide 3D coordinates, but not explicit bond information (or only partial bond information), **OEChem TK** uses the *OEDetermineConnectivity* function. This function deduces the pattern of covalent bonding in a molecule from the proximity of atoms. Two atoms are considered bonded if they are located within the sum of their covalent radii (*OEGetCovalentRadius*) plus an additional "slop" factor of 0.45 Angstroms.

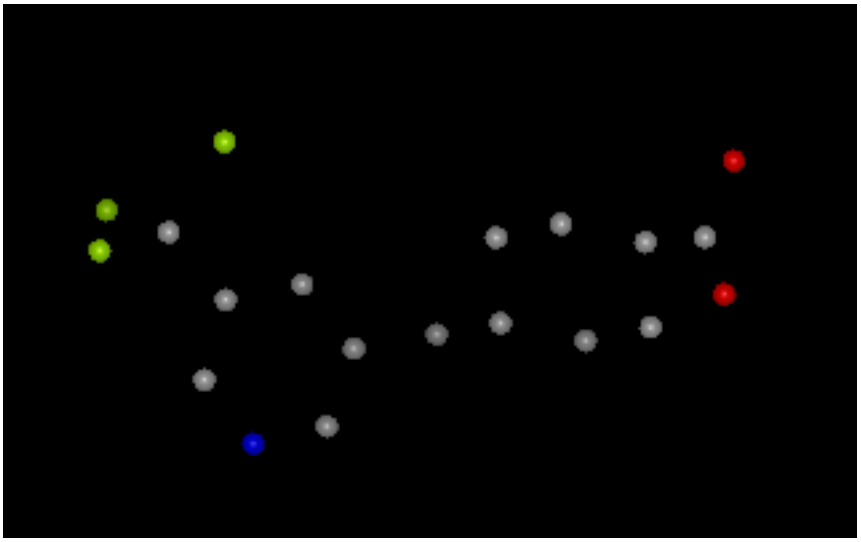


Fig. 7: *Example of 3D molecule with no explicit bond information*

OEDetermineConnectivity will not create a bond between two atoms that are less than 0.4 Angstroms apart. Such unreasonably short bond lengths indicate the structure is either severely distorted, or doesn't have coordinate information at all.

All bonds created by *OEDetermineConnectivity* have bond orders set to one. To perceive bond order information, see **OEChem TK**'s *OEPerceiveBondOrders* function described in the next section.

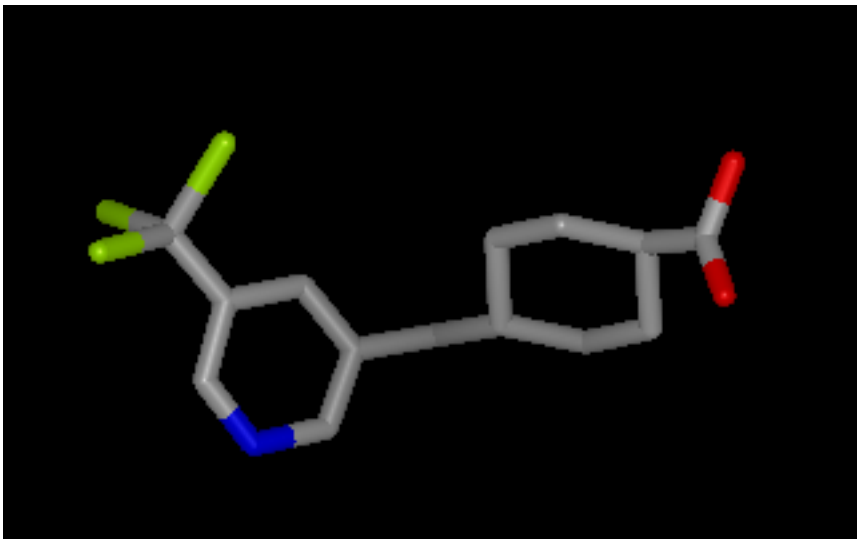


Fig. 8: *Example of 3D molecule with perceived bond connectivity*

The *OEDetermineConnectivity* function checks whether a bond already exists between two atoms before creating a new bond. This allows this function to be used with file formats that may specify partial connectivity, such as only multiple (double, triple or quadruple) bonds.

Bond Order Perception

The *OEPerceiveBondOrders* function is used to deduce bond orders from the 3D coordinates and simple connectivity of a molecule. If the simple connectivity, *i.e.* bonds without bond orders isn't specified in the input file, *OEDetermineConnectivity* should be called first to deduce this information from the 3D coordinates.

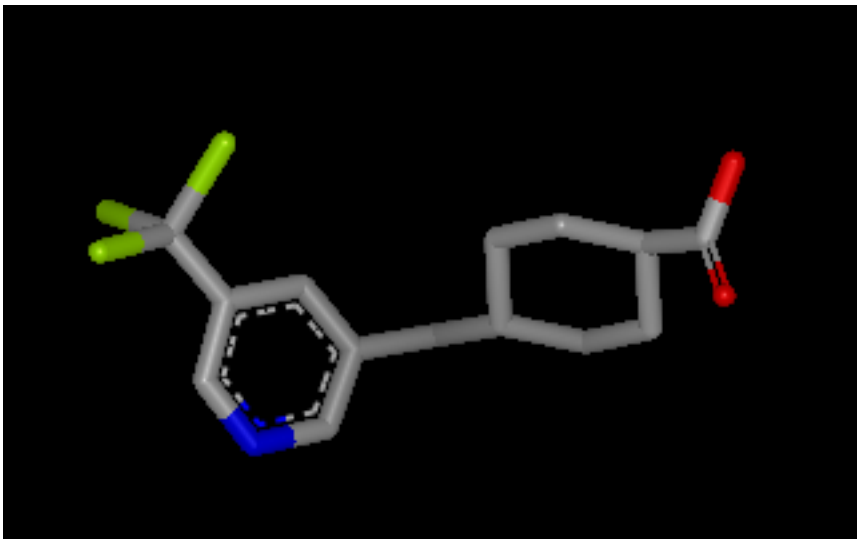


Fig. 9: *Example of 3D molecule with perceived connectivity and bond order*

The following code snippet shows how to perceive connectivity and bond order if a molecule has 3D information but no explicit bond information:

```
OEChem.OEDetermineConnectivity(mol);
OEChem.OEFindRingAtomsAndBonds(mol);
OEChem.OEPerceiveBondOrders(mol);
OEChem.OEAssignImplicitHydrogens(mol);
OEChem.OEAssignFormalCharges(mol);
```

See also

- *OEDetermineConnectivity*
- *OEFindRingAtomsAndBonds*
- *OEPerceiveBondOrders*
- *OEAssignImplicitHydrogens*
- *OEAssignFormalCharges*

Kekulé Form Assignment

A number of file formats don't represent a connection table as a single representative Kekulé form but instead denote some bonds, such as those in benzene, as aromatic. **OEChem TK** provides a method for determining a valid, but arbitrary, Kekulé form for such connection tables using the *OEKekulize* function (see example in *Figure: Kekulization of quinolin*). On input to *OEKekulize*, the integer bond type property of each bond represents either the bond order (1 for single, 2 for double, 3 for triple or 4 for quadruple) or the value 5 indicating the bond is aromatic or resonant. The algorithm sets the bond order property from the bond type property, with the exception of bond type 5, which is assigned a bond order of either 1 or 2 representing either a single or double bond. The boolean return value indicates whether a valid Kekulé form could be assigned.

See also

OEKekulize function is normally only used by low-level file readers for interpreting input connection tables. To write out a Kekulé SMILES string, the aromaticity atom and bond flags have to be cleared by the *OEClearAromaticFlags* function in order to consider a molecule as aliphatic with explicit bond order. See example code in *Clearing Aromaticity*.

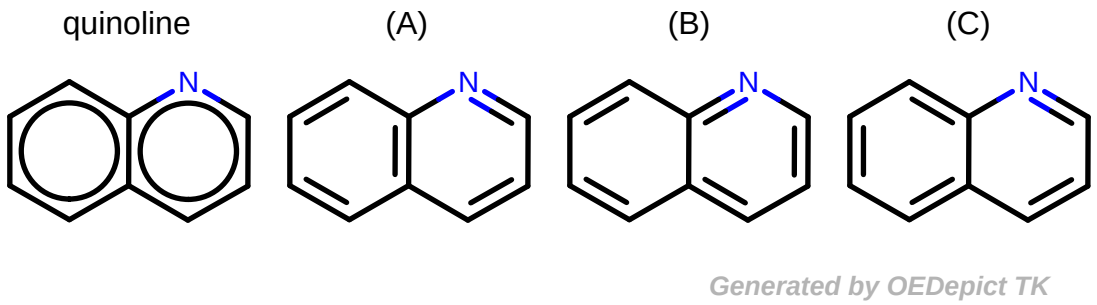


Fig. 10: **Kekulization of quinolin**
OEKekulize() function arbitrary generates one of the three valid Kekulé forms

See also

- [Baber-1992]
- [Hendlich-1997]
- [Kearsley-1993]
- [Labute-1996]

- [Meng-1991]

Kekulé Assignment of New Bonds

Molecules input to *OEKekulize* must have their integer bond type set. The property of each bond represents either the bond order (1 for single, 2 for double, 3 for triple or 4 for quadruple) or the value 5 indicating the bond is aromatic or resonant. When creating new atoms and bonds internally the integer type is set to a default value of 0. The following code snippet outlines how to prepare modified molecules by setting the bond int type so that they can be passed to *OEKekulize*:

```
OEGraphMol mol = new OEGraphMol();
OEChem.OESmilesToMol(mol, "c1ccccc1");
//Code that modifies the molecule
//and creates new atoms and bonds
OEChem.OEFindRingAtomsAndBonds(mol);
OEChem.OEAssignAromaticFlags(mol);
foreach (OEBondBase bond in mol.GetBonds())
{
    if (bond.IsAromatic())
    {
        bond.SetIntType(5);
    }
    else if (bond.GetOrder() != 0)
    {
        bond.SetIntType((int)bond.GetOrder());
    }
    else
    {
        bond.SetIntType(1);
    }
}
OEChem.OEKekulize(mol);
```

2.1.13 Stereochemistry Perception

OEChem TK has the ability to store and retrieve stereochemical information for atoms and bonds independent of two or three dimensional coordinates. The current version of **OEChem TK** supports stereochemistry definitions of *handedness* around tetrahedral centers, and *cis/trans* configuration around bonds.

The table below lists the available **OEChem TK** functions related to stereochemical information.

Name	Description
OE3DToAtomStereo	Set the stereochemistry at the chiral atoms of a molecule, as specified by the molecule's 3D coordinates.
OE3DToBondStereo	Set the stereochemistry at the chiral bonds of a molecule, as specified by the molecule's 3D coordinates.
OE3DToInternalStereo	Assigns the stereochemistry of a molecule from its 3D coordinates.
OEMDLClearBondStereo	Clears the bond stereo fields from all bonds of a molecule.
OEMDLCorrectBondStereo	Corrects the wedge/hash bond assignment of the 2D molecule.
OEMDLHasIncorrectBondSter	Checks whether the wedge/hash bond assignment of the 2D molecule is inconsistent or ambiguous.
OEMDLPerceiveBondStereo	Assigns wedge and hash bonds to a connection table from the **OEChem TK** stereochemistry of each atom of the 2D molecule.
OEMDLStereoFromBondStereo	Sets the **OEChem TK** stereochemistry of each atom of the 2D molecule from the wedge and hash bonds in the connection table.
OEMDLStereoFromParity	Sets the **OEChem TK** stereochemistry for each atom from the MDL stereo parity information.
OEPerceiveCIPStereo	Perceives the Cahn-Ingold-Prelog descriptor for the given atom.
OESetCIPStereo	Sets the internal **OEChem TK** stereochemistry from the given CIP atom stereo descriptor.

Atom Chirality

The following two code snippets demonstrate how to loop over chiral atoms using the *OEIsChiralAtom* functor and the *IsChiral* method of the *OEAtomBase* class.

```
foreach (OEAtomBase atom in mol.GetAtoms(new OEIsChiralAtom()))
{
    Console.WriteLine(atom.GetIdx() + " " + OEChem.OEGetAtomicSymbol(atom.
↪GetAtomicNum()));
}
```

```
foreach (OEAtomBase atom in mol.GetAtoms())
{
    if (atom.IsChiral())
    {
        Console.WriteLine(atom.GetIdx() + " " + OEChem.OEGetAtomicSymbol(atom.
↪GetAtomicNum()));
    }
}
```

Warning

When a molecule is imported from a file, its atom and bond chirality are not automatically perceived, the user has to call *OEPerceiveChiral* explicitly.

Atom Stereochemistry

Stereochemistry around atoms can be set and retrieved by the *OEAtomBase.SetStereo* and *OEAtomBase.GetStereo* methods, respectively. The *OEAtomBase.HasStereoSpecified* method returns a boolean value that indicates whether stereochemical information of a particular type has been stored for an atom. **OEChem TK** currently only supports *OEAtomStereo.Tetrahedral* atom stereochemistry class.

Note that the setting and retrieval of atom stereo requires a vector containing pointers to the neighboring atoms as the first argument. The *handedness* value returned from *OEAtomBase.GetStereo* will depend on the order of the neighboring atoms as they appear in the passed vector.

The first neighbor atom in the vector passed to *OEAtomBase.GetStereo* is taken as atom number one for the determination of *handedness*. Likewise, subsequent atoms in the neighbor atom vector are assigned sequentially to positions in the *handedness* definition. Although, three neighboring atoms are sufficient to determine the *handedness* around a trigonal pyramidal or tetrahedral center, either three or four atoms can be provided to the *OEAtomBase.GetStereo* function when requesting a value for tetrahedral chirality. The following table shows the return values of *OEAtomBase.GetStereo* method by passing the neighbor atoms in different order.

neighbor order	[C@@H](Cl)(Br)N	[C@H](Cl)(Br)N
H Cl Br N	right handed	left handed
H Br Cl N	left handed	right handed
H Cl N Br	left handed	right handed
…	…	…
Cl Br N	left handed	right handed
Br Cl N	right handed	left handed
Cl N Br	right handed	left handed

Note

The definition of *handedness* for tetrahedral stereochemistry does not imply chirality around a tetrahedral center, but rather indicates relative positions of neighboring atoms.

The following code sample demonstrates looping over atoms, testing for atoms that have tetrahedral stereochemistry, and printing out the value of the tetrahedral stereochemistry (either *OEAtomStereo.RightHanded*, *OEAtomStereo.LeftHanded*, or *OEAtomStereo.Undefined*). See also the input structure depicted in *Figure: Atom stereo*.

Listing 1: Accessing atom stereo information

```
using System;
using OpenEye.OEChem;

public class PrintAtomStereo
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "[H][C@]1(NC[C@@H](CC1CO[C@H]2CC[C@@H](CC2)O)N)[H]");

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            bool chiral = atom.IsChiral();
            uint stereo = OEAtomStereo.Undefined;
            if (atom.HasStereoSpecified(OEAtomStereo.Tetrahedral))
            {
                OEAtomBaseVector v = new OEAtomBaseVector();
                foreach (OEAtomBase nbr in atom.GetAtoms())
                {
                    v.Add(nbr);
```

(continues on next page)

(continued from previous page)

```
                }

                stereo = atom.GetStereo(v, OEAtomStereo.Tetrahedral);
            }

            if (chiral || stereo != OEAtomStereo.Undefined)
            {
                Console.Write("Atom: " + atom.GetIdx() + " chiral= " + chiral + "␣
↪stereo= ");
                if (stereo == OEAtomStereo.RightHanded)
                {
                    Console.WriteLine("right handed");
                }
                else if (stereo == OEAtomStereo.LeftHanded)
                {
                    Console.WriteLine("left handed");
                }
                else
                {
                    Console.WriteLine("undefined");
                }
            }
        }
        return 0;
    }
}
```

The output of the preceding program is the following:

```
Atom: 1 chiral=false stereo=left handed
Atom: 4 chiral=true stereo=right handed
Atom: 7 chiral=true stereo=undefined
Atom: 10 chiral=false stereo=left handed
Atom: 14 chiral=false stereo=right handed
```

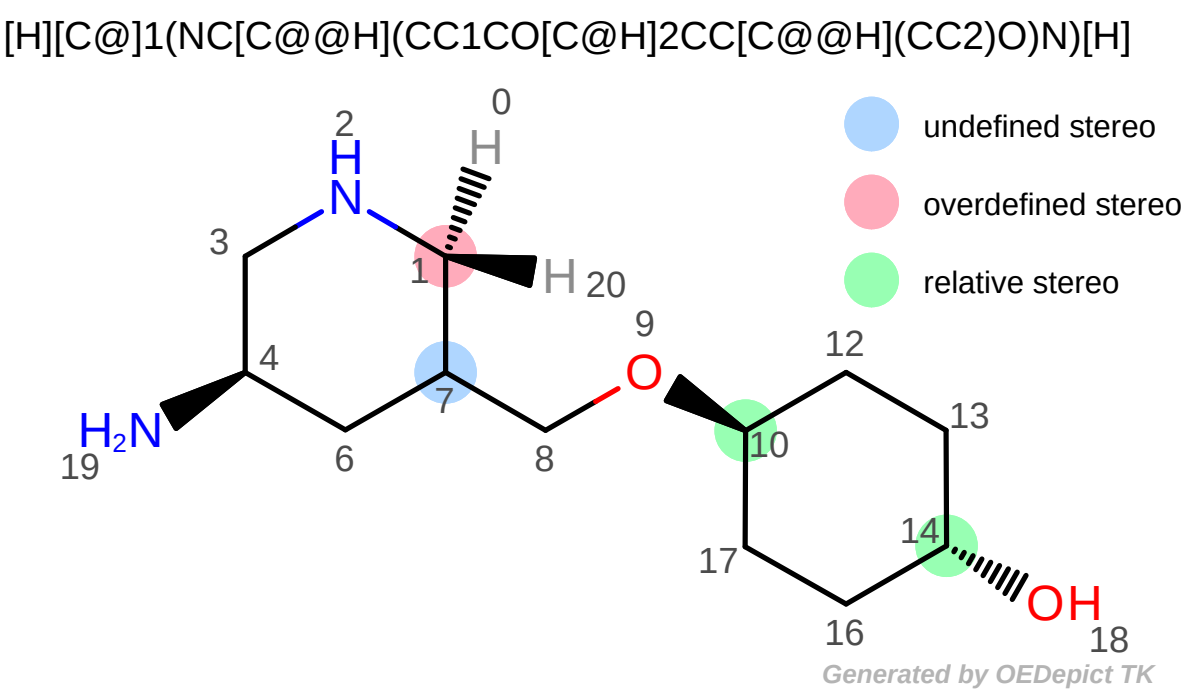


Fig. 11: **Atom stereo**

As mentioned before, tetrahedral stereochemistry (*i.e.* specified atom stereo) does not imply chirality around a tetrahedral center. For example atoms 1, 10 and 14 in *Figure: Atom stereo* have specified atom stereo, even though they are non-chiral. *Vice versa*, atom can be chiral but with unspecified atom stereo configuration (such as atom 7 in *Figure: Atom stereo*). Atoms 1 and 6 are non-chiral but they have atom stereo information and they are relative to each other.

The following example shows how to manually set the atom stereo of atom 7 (chiral, undefined) to *left* by the `OEAtomBase.SetStereo` method.

Listing 2: Setting atom stereo information

```
using System;
using OpenEye.OEChem;

public class SetAtomStereo
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "[H][C@]1(NC[C@@H](CC1CO[C@H]2CC[C@@H](CC2)O)N)[H]");

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            if (atom.IsChiral() && !atom.HasStereoSpecified(OEAtomStereo.Tetrahedral))
            {
                OEAtomBaseVector v = new OEAtomBaseVector();

                foreach (OEAtomBase neigh in atom.GetAtoms())
```

(continues on next page)

(continued from previous page)

```
                {
                    v.Add(neigh);
                }

                atom.SetStereo(v, OEAtomStereo.Tetra, OEAtomStereo.Left);
            }
        }

        Console.Out.WriteLine(OEChem.OEMolToSmiles(mol));
        return 0;
    }
}
```

The output of the preceding program is the following:

```
[H][C@@]1([C@@H](C[C@H](CN1)N)CO[C@H]2CC[C@@H](CC2)O)[H]
```

Just as in *OEAtomBase.GetStereo*, the vector of neighbor atoms provide the references about which the *handedness* is defined. The first of the integer arguments is the stereochemistry type (*i.e.* *OEAtomStereo.Tetrahedral* (or just *OEAtomStereo.Tetra*) , and the second is the associated value (*i.e.* *OEAtomStereo.Right* or *OEAtomStereo.Left*).

Bond Chirality

The following two code snippets demonstrate how to loop over chiral bonds using the *OEIsChiralBond* functor and the *IsChiral* method of the *OEBondBase* class.

```
foreach (OEBondBase bond in mol.GetBonds(new OEIsChiralBond()))
{
    Console.WriteLine(bond.GetBgnIdx() + "=" + bond.GetEndIdx());
}
```

```
foreach (OEBondBase bond in mol.GetBonds())
{
    if (bond.IsChiral())
    {
        Console.WriteLine(bond.GetBgnIdx() + "=" + bond.GetEndIdx());
    }
}
```

Warning

When a molecule is imported from a file, its atom and bond chirality are not automatically perceived, the user has to call *OEPerceiveChiral* explicitly.

Bond Stereochemistry

Stereochemistry around bonds can be specified in a similar fashion to stereochemistry about atom centers. The request for stereochemistry of a particular class can be made of an *OEBondBase* using the method *OEBondBase.HasStereoSpecified*. **OEChem TK** currently only supports *OEBondStereo.CisTrans* bond stereochemistry class that defines whether the configuration of two atoms around a non-rotatable bond is *cis* or *trans*. If a bond has associated stereochemical data, it can be retrieved using *OEBondBase.GetStereo* method.

The following code sample demonstrates a loop over bonds that tests for bonds with associated stereochemistry, and retrieval of whether the neighboring atoms are *cis* or *trans* relative to one another. (See also the input structure depicted in *Figure: Bond stereo*.

Listing 3: Accessing bond stereo information

```
using System;
using OpenEye.OEChem;

public class PrintBondStereo
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "C\\C=C\\C=C/C=CC");

        foreach (OEBondBase bond in mol.GetBonds())
        {
            if (bond.HasStereoSpecified(OEBondStereo.CisTrans))
            {
                foreach (OEAtomBase atom1 in bond.GetBgn().GetAtoms())
                {
                    if (atom1.GetIdx() == bond.GetEnd().GetIdx())
                    {
                        continue;
                    }

                    foreach (OEAtomBase atom2 in bond.GetEnd().GetAtoms())
                    {
                        if (atom2.GetIdx() == bond.GetBgn().GetIdx())
                        {
                            continue;
                        }

                        OEAtomBaseVector v = new OEAtomBaseVector();
                        v.Add(atom1);
                        v.Add(atom2);
                        uint stereo = bond.GetStereo(v, OEBondStereo.CisTrans);
                        Console.Write("Atoms: " + atom1.GetIdx() + " and " + atom2.
↪GetIdx() + " are ");
                        if (stereo == OEBondStereo.Cis)
                        {
                            Console.WriteLine("cis");
                        }
                        else if (stereo == OEBondStereo.Trans)
                        {
                            Console.WriteLine("trans");
                        }
                    }
                }
            }
        }
        return 0;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

The output of the preceding program is the following:

```
Atoms: 0 and 3 are trans
Atoms: 2 and 5 are cis
```

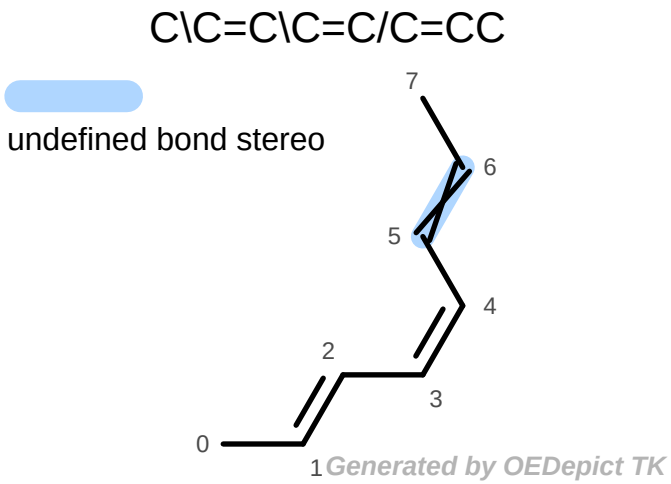


Fig. 12: **Bond stereo**

Note that even though the double bond defined by atoms 4–5–6–7 is depicted as *trans* (see *Figure: Bond stereo*) it is still considered undefined, since no stereo information is encoded for this bond in the input SMILES.

The following example shows how to manually set this missing bond stereo to *trans* by the *OEBondBase.SetStereo* method.

Listing 4: Setting bond stereo information

```
using System;
using OpenEye.OEChem;

public class SetBondStereo
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "C\\C=C\\C=C/C=CC");

        foreach (OEBondBase bond in mol.GetBonds())
        {
            if (bond.GetOrder() == 2 && !bond.HasStereoSpecified(OEBondStereo.CisTrans))
            {
                OEAtomBaseVector v = new OEAtomBaseVector();
                foreach (OEAtomBase neigh in bond.GetBgn().GetAtoms())
                {
```

(continues on next page)

(continued from previous page)

```
                    if (neigh.GetIdx() != bond.GetEnd().GetIdx())
                    {
                        v.Add(neigh);
                        break;
                    }
                }

                foreach (OEAtomBase neigh in bond.GetEnd().GetAtoms())
                {
                    if (neigh.GetIdx() != bond.GetBgn().GetIdx())
                    {
                        v.Add(neigh);
                        break;
                    }
                }

                bond.SetStereo(v, OEBondStereo.CisTrans, OEBondStereo.Trans);
            }
        }

        Console.WriteLine(OEChem.OEMolToSmiles(mol));
    }
}
```

The first integer argument of the method *OEBondBase.SetStereo* is used to specify the type of stereochemistry (*i.e.* *OEBondStereo.CisTrans*). The second integer argument specifies the bond stereo configuration (*i.e.* *OEBondStereo.Cis* or *OEBondStereo.Trans*)

The output of the preceding program is the following:

```
C/C=C/C=C\C=C\C
```

CIP Stereo Perception

The following two examples demonstrate how to perceive CIP (Cahn–Ingold–Prelog) atom and bond stereo information from the molecules created by the SMILES representation. If the input molecule has 3D coordinates, the *OE3DToInternalStereo* function should be called.

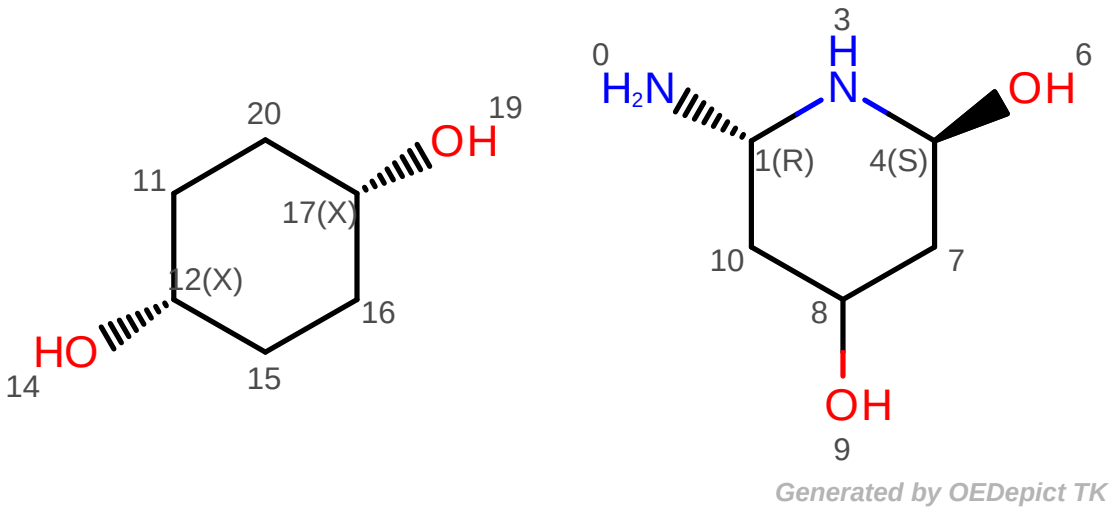


Fig. 13: **CIP atom stereo**

Listing 5: Perceiving CIP atom stereo information

```
using System;
using OpenEye.OEChem;

public class PrintCIPAtomStereo
{
    public static void Main(string [] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "N[C@@H]1N[C@@H](O)CC(O)C1.C2[C@@H](O)CC[C@@H](O)C2");

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            uint cip = OEChem.OEPerceiveCIPStereo(mol,atom);

            if (atom.HasStereoSpecified())
            {
                Console.Write("atom " + atom.GetIdx() + " is ");

                if (cip == OECIPAtomStereo.S)
                {
                    Console.WriteLine("S");
                }
                else if (cip == OECIPAtomStereo.R)
                {
                    Console.WriteLine("R");
                }
                else if (cip == OECIPAtomStereo.NotStereo)
                {
```

(continues on next page)

(continued from previous page)

```
                    Console.WriteLine("not a CIP stereo center");
                }

            }

            if (cip == OECIPAtomStereo.UnspecStereo)
            {
                Console.Write("atom " + atom.GetIdx() + " is ");
                Console.WriteLine("a CIP stereo center without specified stereochemistry
↪");
            }
        }
    }
}
```

The *OEPerceiveCIPStereo* function detects the Cahn-Ingold-Prelog descriptor of the given atom and returns a value from the *OECIPAtomStereo* namespace.

The output of *Listing 5* is the following: (See also the input structure depicted in *Figure: CIP atom stereo*.

```
atom 1 is R
atom 4 is S
atom 8 is a CIP stereo center without specified stereochemistry
atom 12 is not a CIP stereo center
atom 17 is not a CIP stereo center
```

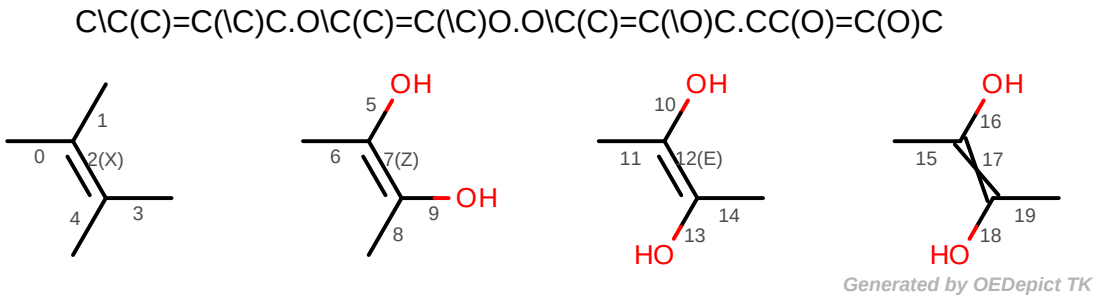


Fig. 14: **CIP bond stereo**

Listing 6: Perceiving CIP bond stereo information

```
using System;
using OpenEye.OEChem;

public class PrintCIPBondStereo
{
    public static void Main(string [] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "C\\C(C)=C(\\C)C.O\\C(C)=C(\\C)O.O\\C(C)=C(\\O)C.
↪CC(O)=C(O)C");
```

(continues on next page)

(continued from previous page)

```
        foreach (OEBondBase bond in mol.GetBonds())
        {
            if (bond.GetOrder() == 2)
            {

                uint cip = OEChem.OEPerceiveCIPStereo(mol,bond);

                if (bond.HasStereoSpecified())
                {
                    Console.Write("bond " + bond.GetIdx() + " is ");
                    if (cip == OECIPBondStereo.E)
                    {
                        Console.WriteLine("E");
                    }
                    else if (cip == OECIPBondStereo.Z)
                    {
                        Console.WriteLine("Z");
                    }
                    else if (cip == OECIPBondStereo.NotStereo)
                    {
                        Console.WriteLine("not a CIP stereo center");
                    }
                }

                if (cip == OECIPBondStereo.UnspecStereo)
                {
                    Console.Write("bond " + bond.GetIdx() + " is ");
                    Console.WriteLine("a CIP stereo center without specified␣
↪stereochemistry");
                }
            }
        }
    }
}
```

The *OEPerceiveCIPStereo* function detects the Cahn-Ingold-Prelog descriptor of the given bond and returns a value from the *OECIPBondStereo* namespace.

The output of *Listing 6* is the following: (See also the input structure depicted in *Figure: CIP bond stereo*.

```
bond 2 is not a CIP stereo center
bond 7 is Z
bond 12 is E
bond 17 is a CIP stereo center without specified stereochemistry
```

2.1.14 Formal and Partial Charges

Each *OEAtomBase* keeps track of two types of charges:

- The first, **formal charge**, is an integer property that is essential for the correct valence representation of a molecule. Together with atomic valences, bond order and the connectivity, this field defines the identity of a molecule.

- The second type of charge, **partial charge**, is a floating point property used in computational chemistry and molecular modeling. This value is used to represent the electronic distribution/wave-function of a molecule by approximating the molecule's electrostatic field with a set of point charges located at each atom.

Property Name	Type	Get Method	Set Method
Formal Charge	integer	*GetFormalCharge*	*SetFormalCharge*
Partial Charge	double	*GetPartialCharge*	*SetPartialCharge*

Neither the formal charge nor the partial charge is a directly observable property of an atom. Instead the same molecule may be represented by different valence representations, each placing the formal charges in different locations, *i.e.* different partial charging algorithms may assign significantly different partial charges to the same atom (for example benzene can be represented as `[cH+]1[cH-][cH+][cH-][cH+][cH-]1`)

Formal Charges Assignment

Normally, file formats such as SMILES, SLN or MDL's SDF format, specify the formal charges on each atom of a connection table. However, when reading from other file formats or when repairing 'broken' molecules, it may be convenient to assign formal charges to each atom such that the atomic valence is consistent. **OEChem TK** provides this functionality via the *OEAssignFormalCharges* function. This function requires that bond orders and implicit hydrogen counts have been set on a molecule. It then adjusts the formal charge on each uncharged atom to correct common valence model mismatches. For example, quaternary nitrogens are assigned a +1 formal charge, and terminal oxygen connected only by a single bond (with no implicit hydrogens) is assigned a -1 formal charge.

A more technical discussion of the formal charges that are assigned by this function are described in the *OpenEye Charge Model* section.

Gasteiger Partial Charge Calculation

To assign Marsilli-Gasteiger partial charges to a molecule, **OEChem TK** provides the *OEGasteigerPartialCharges* function. This sets the partial charge property of each atom, using the *SetPartialCharge* method. The algorithm itself reproduces the partial charges as calculated by Tripos Inc's Sybyl software, with default parameter settings, which is the *de facto* reference implementation for Gasteiger charges.

The Gasteiger partial charge algorithm currently assumes that all hydrogen atoms are represented explicitly, for example by calling *OEAddExplicitHydrogens*.

The current version of *OEGasteigerPartialCharges* should return the same results independent of the currently assigned aromaticity models and the values of each atom's "integer atom type" property. Early versions of **OEChem TK** allowed customization of the Gasteiger charge calculation by explicitly assigning the Tripos atoms types of each atom of the molecule. Unfortunately, this required assigning Tripos' aromaticity and assigning Tripos atom types before each call to *OEGasteigerPartialCharges*. The more recent behavior is less error-prone (as it's no longer possible to forget to prepare a molecule) and greatly simplifies common usage.

The first stage of the Marsilli-Gasteiger *"Partial Equalization of Orbital Electronegativities"* (PEOE)" calculation [Gasteiger-1980] is assignment of seed charges to each atom. Typically for neutral atoms the partial charge seed is zero, but for example, each oxygen in a carboxylate is assigned -0.5, and the net formal charge on conjugated rings is shared equally amongst the atoms of the ring system. These seed charges may also be useful in some applications, and can be assigned using the *OEGasteigerInitialCharges* method.

See also

- [Gasteiger-1978]
- [Gasteiger-1980]

2.1.15 2D Coordinate Generation

The *OEGenerate2DCoordinates* function **OEChem TK**'s depiction algorithm is capable of assigning 2D coordinates to molecules and reactions. See *Figure: Schematic representation of the steps of the 2D coordinate generation*. The algorithm uses a **built-in** ring template dictionary to provide aesthetically pleasing 2D layouts for complex macrocycles. In case when no built-in template exists, the 2D coordinates of the ring system are generated on-the-fly. After laying out the ring system(s), the algorithm assigns coordinates for acyclic chains and functional groups. In the last step, the algorithm attempts to resolve atom clashes using a variety of heuristics.

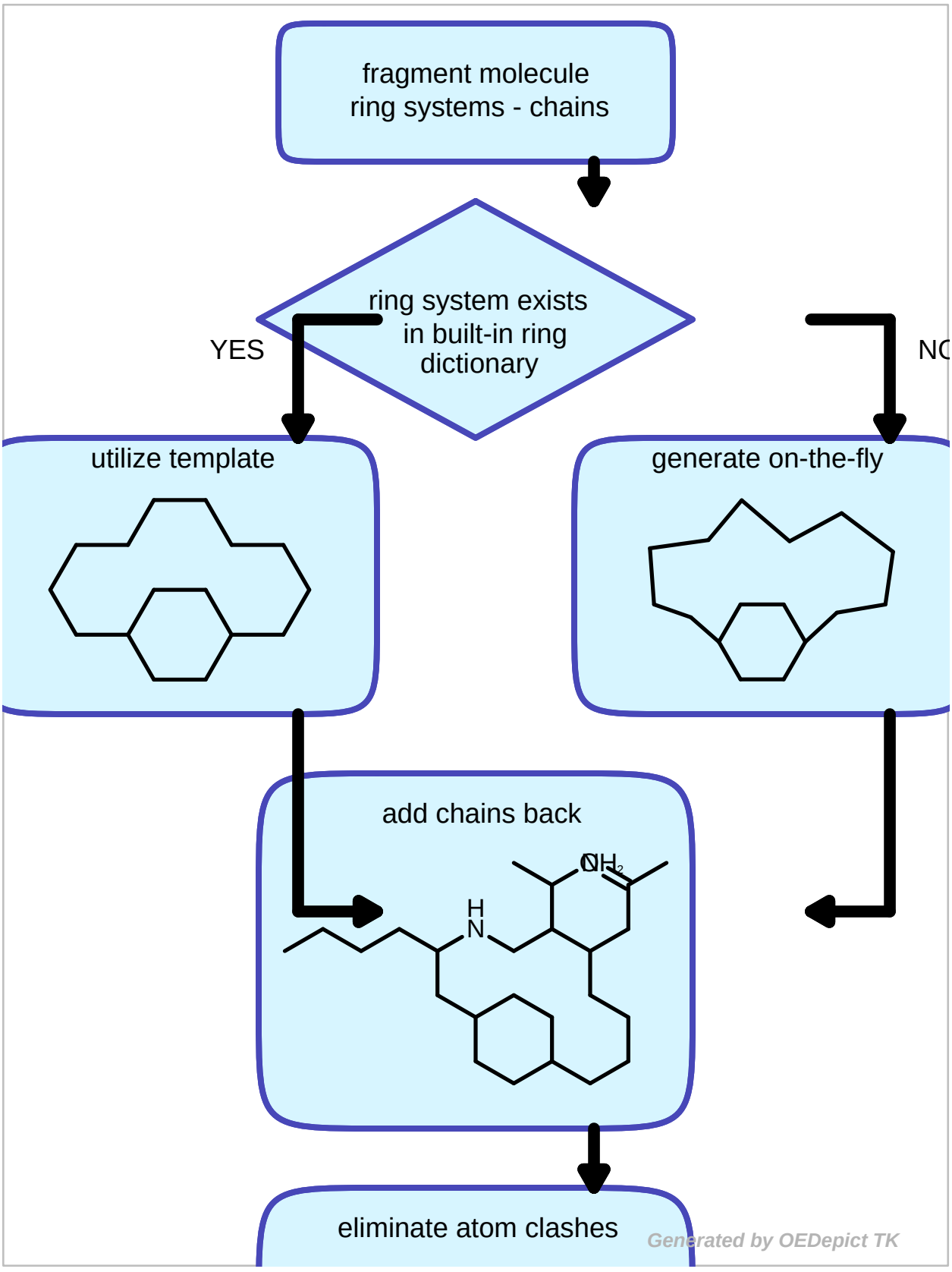


Fig. 15: **Schematic representation of the steps of the 2D coordinate generation in OEChem TK**

This built-in dictionary currently contains about 3000 ring-systems that frequently occur in drug-like databases. New ring templates are continuously being added with each release.

Note

To improve the depiction of the ring systems, please run the *Extract 2D ring templates* script on a 2D molecule file and send the results to support@eyesopen.com. **OpenEye** will be happy to incorporate the new ring systems in the **built-in** dictionary in the next release of the toolkits.

Note

`OEGenerate2DCoordinates` is a **low-level** function that only assigns 2D coordinates to the given molecule. It is highly recommended to use the `OEPrepareDepiction` function of **OEDepict TK** when generating 2D coordinates for molecular depiction. The `OEPrepareDepiction` function, apart from assigning 2D coordinates also does the following:

- suppresses or adds explicit hydrogens
- perceives atom and bond stereo information
- optimizes the 2D coordinates for horizontal or vertical rendering

User-Defined Ring Dictionary

OEChem TK also allows the customization of the 2D coordinate generation process by providing functionality to extract ring templates from 2D molecule files, build user-defined ring dictionaries, and then plug them into the 2D coordinate generation engine.

The following are the steps to define your own ring dictionaries:

1. Building a ring dictionary from scratch

   See also

   Example code : *Create 2D ring dictionary*

2. Adding new ring templates to an existing ring dictionary

   See also

   Example code : *Append 2D ring dictionary*

3. Generating 2D coordinates with custom ring templates

   See also

   Example code : *Generate 2D coordinates with user-defined ring templates*

See also

- *OE2DRingDictionary* class
- *OE2DRingDictionaryCreatorOptions* class
- *OEInit2DRingDictionary* function
- *OEIsValid2DRingDictionary* function
- *OEWrite2DRingDictionary* function

2.1.16 Molecule Styling

OE3DMolStyle

An *OEMolBase* has a default style without the addition of an *OE3DMolStyle*. The *OE3DMolStyle* class gives a multitude of customization options for the *OEMolBase*, from coloring to labeling. Outside of *OEMolBase*, *OE3DMolStyle* can be applied to an *OEAtomBase* and an *OEConfBase*. There is a hierarchy when applying *OE3DMolStyle* to each one of these classes: *OEMolBase* -> *OEConfBase* -> *OEAtomBase*.

Atom Labeling

Preset labels can be applied using *SetAtomLabelType*. Custom labels can be applied to specific atoms utilizing *SetAtomLabelText*. *SetAtomLabelType* takes in *OEAtomLabelScheme*. The example below depicts the B-Factor above each one of the respective atoms in the molecule.

Styling and Typing

There are multiple styling and typing options for an *OE3DMolStyle*. *OEAtomStyle* can be set using *SetAtomStyle*. This allows for customization of the visibility of the atoms in a molecule, which ranges from hidden atoms to CPK atom styling. In the example below the atoms are set to display as Ball and Stick.

OEProteinStyle can be set utilizing *SetProteinStyle*. This allows for customization of the protein from hidden to ribbon. The example below set the protein style to CAlpha, which depicts the protein with edges at the CAlphas.

OEContourStyle can be set utilizing *SetContourStyle*. This allows for customization of the contour from hidden to solid. *SetGridType* takes in the *OEStyleGridType* constant to specify the type of the grid being depicted. The example below set the grid style to mesh and the drid type to electrostatic. This will depict the grid in mesh form for better visibility of the other components of the molecule.

SetSurfaceStyle utilizes *OESurfaceStyle* to depict the surface by mesh, point, and solid. *OESurfaceType* allows for surface typing when set by *SetSurfaceType*. The example below style a molecular surface to be depicted in points.

Hydrogen visibility is customizable by utilizing *OEHydrogenVisibility* in *SetHydrogenVisibility*. This allows for all hydrogen, polar hydrogen, and hidden hydrogen depictions. In the example below sets up an *OE3DMolStyle* to only depict the polar hydrogens in the molecule.

Atom to atom interactions is depicted using arrows from one atom to another. By default this is set to off. *SetInteractionStyle* utilizes *OEInteractionStyle* to toggle between intermolecular hydrogen bond interaction depiction, intramolecular hydrogen bond interaction depiction, and no bond interaction depiction. In this example we set the the interaction style to intramolecular hydrogen bond depiction.

Colorer

Multple aspects of the molecule can be colored when *OE3DMolStyle* takes advantage of *OEMolStyleColorer*. There are preset color schemes for all parts of the molecule from atoms to protein, but custom colors can also be added for better personalization of the molecule.

OEAtomColorScheme provides multple preset atom color schemes when specified in the construction of an *OEMolStyleColorer*. *SetAtomColorer* takes in the *OEMolStyleColorer*. Atoms are colored by element by default, but this allows for coloring by b factor, residues, formal charge, and other aspects of the atoms and/or molecule. The labels of an atom can also be colored when an *OEMolStyleColorer* with a default color and/or custom colors by atom name or atomic number. The color of a grid can also be set with a default color specified in an *OEMolStyleColorer*. The *OEMolStyleColorer* with the default color can be set to the *OE3DMolStyle* with *SetGridContourColorer*.

OESurfaceColorScheme provides customization of surfaces with preset color schemes. These color schemes are applied similarly to how atom color schemes are applied but by utilizing *SetSurfaceColorer*. When a color scheme isn't specified the surface is colored by the default color, by default the surface is colored blue.

Proteins can be colored utilizing a color scheme from *OEProteinColorScheme* but it can also colored with a constant from *OEAtomColorScheme*. The *OEMolStyleColorer* utilized for the protein can be set to the *OE3DMolStyle* with *SetProteinColorer*. The example below adds the different coloring options to the molecule. The protein is colored by residue. The atoms are colored by element, with a default color of white, but the *OEMolStyleColorer* also overwrites preset colors of certain atoms by atomic number. The grid is colored pink and the surface is colored by surface potential.

See also

- *basicStyle* example
- *bFactorStyle* example
- *styleHierarchy* example

Molecule Algorithms

2.1.17 Pattern Matching

OEChem TK includes facilities to perform different types of pattern (graph) matching. Graph matching is based on node (atom) and edge (bond) correspondences. An atom which satisfies the conditions of a node in a query graph is said to match. Likewise, a bond which satisfies the conditions of an edge in a query graph is said to match. Pattern matching is the process of identifying groupings of matching nodes and edges. Substructure search, or subgraph isomorphism, is the process of finding a graph match which is less than or equal to a larger graph. Maximum common substructure search is the process of identifying the maximal graph correspondence between two graphs. Clique detection is the process of finding all possible correspondences between two graphs within a set of bounds.

Substructure Search

Substructure searches can be done in **OEChem TK** using the *OESubSearch* class. The *OESubSearch* class can be initialized with a *SMARTS* pattern, an *OEQMolBase* query molecule, or an *OEMolBase* with expression options. The following example demonstrates how to initialize a *OESubSearch* instance with a *SMARTS* pattern, and perform a substructure search.

Listing 1: Simple substructure search example

```
using System;
using OpenEye.OEChem;

public class SubstructureSearch
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccccc1C");
```

(continues on next page)

(continued from previous page)

```
        /* create a substructure search object */
        OESubSearch ss = new OESubSearch("c1ccccc1");
        OEChem.OEPrepareSearch(mol, ss);

        if (ss.SingleMatch(mol))
        {
            Console.WriteLine("benzene matches toluene");
        }
        else
        {
            Console.WriteLine("benzene does not match toluene");
        }
    }
}
```

In *Listing 1*, the query pattern is benzene and the molecule in which the substructure is being searched for is toluene. Since benzene is a substructure of toluene the *OESubSearch.SingleMatch* method will return `true`. The *SingleMatch* method returns `true` if a single subgraph isomorphism is detected in the molecule passed as the function argument.

Using the *SingleMatch* function in *Listing 2*, it is possible to loop through any molecular file and extract molecules that match a particular *SMARTS* pattern.

Listing 2: Extracting molecules based upon a substructure

```
using System;
using OpenEye.OEChem;

public class MolFind
{
    public static int Main(string[] args)
    {
        OESubSearch ss   = new OESubSearch(args[0]);

        oemolistream ifs = new oemolistream(args[1]);
        oemolostream ofs = new oemolostream(args[2]);

        OEGraphMol mol = new OEGraphMol();

        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEPrepareSearch(mol, ss);
            if (ss.SingleMatch(mol))
            {
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }
        return 0;
    }
}
```

The *OESubSearch* class is able to identify the atom and bond correspondences of the pattern and target structures. *Listing 3* extends the simple match example to write out all atom correspondences between benzene and toluene.

Listing 3: Substructure search with atom mappings

```
using System;
using OpenEye.OEChem;

public class AtomMap
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccccc1C");

        OESubSearch ss = new OESubSearch("c1ccccc1");
        OEChem.OEPrepareSearch(mol, ss);

        int count = 1;
        /* loop over matches */
        foreach (OEMatchBase match in ss.Match(mol))
        {
            Console.Write("Match " + count + " : ");
            Console.Write("pattern atoms: ");

            foreach (OEMatchPairAtom ma in match.GetAtoms())
            {
                Console.Write(ma.pattern.GetIdx() + " ");
            }

            Console.Write("target atoms:");
            foreach (OEMatchPairAtom ma in match.GetAtoms())
            {
                Console.Write(" " + ma.target.GetIdx());
            }

            Console.WriteLine();
            count++;
        }
        return 0;
    }
}
```

The output of *Listing 3* is the following:

```
Match 1 : pattern atoms: 0 1 2 3 4 5 target atoms:  0 1 2 3 4 5
Match 2 : pattern atoms: 0 1 2 3 4 5 target atoms:  0 5 4 3 2 1
Match 3 : pattern atoms: 0 1 2 3 4 5 target atoms:  1 2 3 4 5 0
Match 4 : pattern atoms: 0 1 2 3 4 5 target atoms:  1 0 5 4 3 2
Match 5 : pattern atoms: 0 1 2 3 4 5 target atoms:  2 3 4 5 0 1
Match 6 : pattern atoms: 0 1 2 3 4 5 target atoms:  2 1 0 5 4 3
Match 7 : pattern atoms: 0 1 2 3 4 5 target atoms:  3 4 5 0 1 2
Match 8 : pattern atoms: 0 1 2 3 4 5 target atoms:  3 2 1 0 5 4
Match 9 : pattern atoms: 0 1 2 3 4 5 target atoms:  4 5 0 1 2 3
Match 10 : pattern atoms: 0 1 2 3 4 5 target atoms:  4 3 2 1 0 5
Match 11 : pattern atoms: 0 1 2 3 4 5 target atoms:  5 0 1 2 3 4
Match 12 : pattern atoms: 0 1 2 3 4 5 target atoms:  5 4 3 2 1 0
```

The *OESubSearch.Match* method performs subgraph isomorphism determination for instances of *OEMolBase* or *OEQMolBase* and returns an iterator over all detected subgraphs. Each of the detected subgraphs can be queried for their atom and bond correspondences. In this particular example, the benzene substructure is identified twelve times in toluene. There are twelve matches because the benzene ring can be rotated around for 6 matches, and then flipped and rotated around for another 6 matches, yielding a total of 12 matches. Each of the matches differ in their atom and bond correspondences to the pattern substructure.

A match or subgraph is considered unique if it differs from all other subgraphs found previously by at least one atom. When doing unique matching, two subgraph matches which cover the same atoms, albeit in different orders, will be called duplicates and it will be discarded. In order to retrieve only unique matches, the *Match* method has to be called with a second argument being set to `true`. In *Listing 3*, using unique search would yield only a single match for benzene in toluene.

An *OESubSearch* may be initialized using a *SMARTS* or a query molecule *OEQMolBase*. Query molecules must have atom and bond expressions built for the entire molecule to be able to initialize the search object (see *OEQMolBase.BuildExpressions* in the API).

OESubSearch.GetPattern returns a read-only reference to the query molecule contained in an instance of *OESubSearch*. *OEQMolBase* methods can be used to interrogate the returned *OEQMolBase* reference.

The *OESubSearch.SetMaxMatches* method sets the maximum number of subgraphs to be returned by the *OESubSearch.Match* method. Once the maximum number of subgraphs has been found the search is terminated. By default, an *OESubSearch* is constructed with the maximum number of matches set to `1024`. The constraint on the maximum number of matches can be removed by calling *OESubSearch.SetMaxMatches* with a value of `0`.

When an *OESubSearch* object is initialized with a query molecule, the *OESubSearch.Match* method returns a match between the copy of the query molecule stored inside the *OESubSearch* object and the target molecule.

The *Listing 4* example below illustrates how to reconstruct matches between the original query molecule that is used to initialize a search and a target molecule.

In order to keep track of the atoms, each atom of the query molecule is marked with its own unique atom index before the *OESubSearch* object is constructed. After the matches are returned by calling the *Match* the original atoms can be retrieved by these indices.

Listing 1: Substructures match for query molecule

```
       uint itag = OEChem.OEGetTag("__orig_idx");
       foreach (OEAtomBase ai in qmol.GetAtoms())
           ai.SetIntData(itag, (int) ai.GetIdx());

       OESubSearch ss = new OESubSearch(qmol, OEExprOpts.DefaultAtoms, OEExprOpts.
↪DefaultBonds);

       OEGraphMol tmol = new OEGraphMol();
       OEChem.OESmilesToMol(tmol, "Cc1ccccc1");
       OEChem.OEPrepareSearch(tmol, ss);

       foreach (OEMatchBase mi in ss.Match(tmol, true))
       {
         OEMatch match = new OEMatch();
         foreach (OEMatchPairAtom apair in mi.GetAtoms())
         {
           uint pidx = (uint) apair.pattern.GetIntData(itag);
           OEAtomBase pattern = qmol.GetAtom(new OEHasAtomIdx(pidx));
           match.AddPair(pattern, apair.target);
```

(continues on next page)

(continued from previous page)

```
        }
    }
```

See also

- [Figueras-1972]
- [Ullman-1976]

Maximum Common Substructure Search

The maximum common substructure (henceforth MCS) of two molecular graphs can be identified using the *OEMCSSearch* class. *Listing 5* demonstrates how to initialize an *OEMCSSearch* object, perform a maximum common substructure search, and then retrieve the matches.

Listing 5: Maximum common substructure search

```
using System;
using OpenEye.OEChem;

public class MaximumCommonSS
{
    public static int Main(string[] args)
    {
        OEGraphMol pattern = new OEGraphMol();
        OEChem.OESmilesToMol(pattern, "c1cc(O)c(O)cc1CCN");
        OEGraphMol target = new OEGraphMol();
        OEChem.OESmilesToMol(target,  "c1c(O)c(O)c(Cl)cc1CCCBr");

        uint atomexpr = OEExprOpts.DefaultAtoms;
        uint bondexpr = OEExprOpts.DefaultBonds;
        /* create maximum common substructure object */
        uint mcsstype = OEMCSType.Exhaustive;
        OEMCSSearch mcss = new OEMCSSearch(pattern, atomexpr, bondexpr, mcsstype);
        /* set scoring function */
        mcss.SetMCSFunc(new OEMCSMaxAtoms());
        /* ignore matches smaller than 6 atoms */
        mcss.SetMinAtoms(6);

        bool unique = true;
        int count = 1;
        /* loop over matches */
        foreach (OEMatchBase match in mcss.Match(target, unique))
        {
            System.Console.WriteLine("Match " + count + ":");

            System.Console.Write("pattern atoms: ");
            foreach (OEMatchPairAtom ma in match.GetAtoms())
            {
                System.Console.Write(ma.pattern.GetIdx() + " ");
            }
```

(continues on next page)

(continued from previous page)

```
            System.Console.WriteLine();

            System.Console.Write("target atoms:  ");

            foreach (OEMatchPairAtom ma in match.GetAtoms())
            {
                System.Console.Write(ma.target.GetIdx() + " ");
            }

            System.Console.WriteLine();

            // create match subgraph
            OEGraphMol m = new OEGraphMol();
            OEChem.OESubsetMol(m, match, true);
            System.Console.WriteLine("match smiles = " + OEChem.OEMolToSmiles(m));
            count++;
        }
        return 0;
    }
}
```

The first molecule, `pattern`, is dopamine, and the second molecule, `target`, is a dopamine analog. The *OEMCSSearch* instance is initialized with the dopamine query, atom and bond expressions, and the type of the search.

```
uint mcsstype = OEMCSType.Exhaustive;
OEMCSSearch mcss = new OEMCSSearch(pattern, atomexpr, bondexpr, mcsstype);
```

The atom and bond expressions define criteria for atom and bond equivalence used during the search and are defined in the *OEExprOpts* namespace. See the *OEExprOpts Namespace* section for more information.

An *OEMCSSearch* object can also be constructed from a *SMARTS* string directly. In this case standard SMARTS matching rules apply for what constitutes a match. If it is constructed with an *OEQMolBase*, then whatever atom and bond expressions have been applied to the *OEQMolBase* will apply in the MCS search.

The last argument of the initialization defines the search type, either *Approximate* or *Exhaustive*. The difference between the two search types is detailed in the *Exhaustive and approximate MCSS* section. This argument is optional, if it is not specified, then the *exhaustive* method is employed.

During the search process, each identified common substructure is evaluated by a scoring function and only substructures with the best score are retained. The *OEMCSSearch.SetMCSFunc* method provides an ability to set the scoring function of an *OEMCSSearch* object, thereby influencing the result of the maximum common substructure search process. See the *MCS scoring functions* section for more information.

```
mcss.SetMCSFunc(new OEMCSMaxAtoms());
```

The *OEMCSSearch.Match* method returns an iterator over the maximum common substructures.

```
foreach (OEMatchBase match in mcss.Match(target, unique))
{
```

The *OEMatchBase* is then passed as an argument to the *OESubsetMol* function, and subsequently converted into a SMILES string.

```
OEChem.OESubsetMol(m, match, true);
```

The detected maximum common substructure of the example program is depicted in *Figure: Example of MCSS*. The following is the output of the program in *Listing 5*:

```
Match 1:
pattern atoms: 0 1 2 3 4 5 6 7 8 9
target atoms:  7 5 3 4 1 2 0 8 9 10
match smiles = CCc1ccc(c(c1)O)O
```

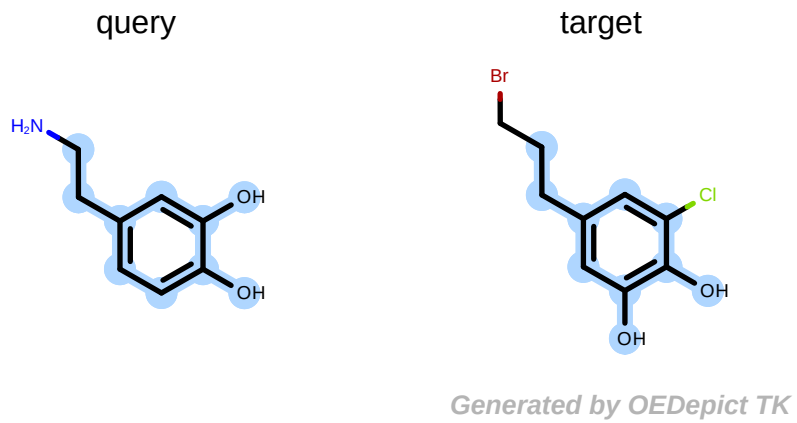


Fig. 16: **Example of MCSS**
The maximum common substructure (highlighted by light blue) of dopamine and dopamine analog.

The maximum common substructure search can perform unique or non-unique substructure searching by changing the second argument of the *OEMCSSearch.Match* method. The default is a non-unique search.

By definition, a match or subgraph is considered unique if it differs from all other subgraphs found previously by at least one atom or bond. Additionally, it is also considered unique if the pattern subgraph is mapped to a different part of the target.

Figure: Unique match shows an example in which the two matches are identified using the unique search method. Even though the two obtained subgraphs are identical, they represent different mappings between the pattern and the target, therefore they are both considered unique. Using a non-unique search would result in four matches, since the phenol can flip, yielding two additional matches.

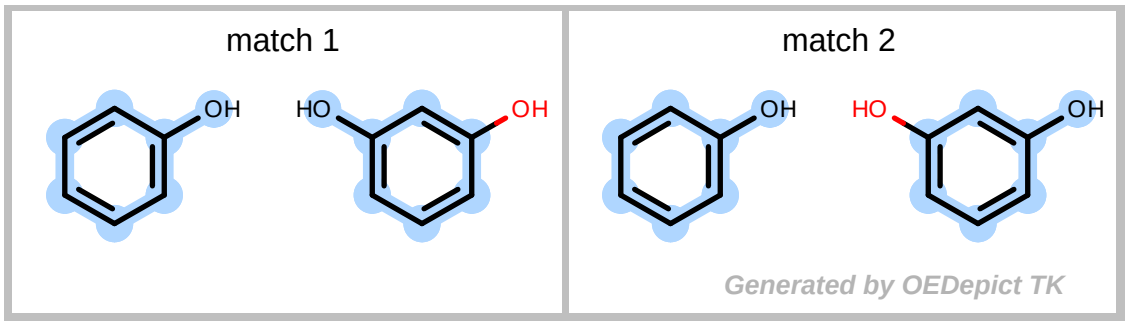


Fig. 17: **Unique match**
Example for unique maximum common substructures

The search space of a maximum common subgraph determination can be restricted by constraining pairs of atoms or bonds to be mapped onto one another in all subgraph solutions. This is done using the *OEMCSSearch.AddConstraint* method. Failure to satisfy atom or bond pairwise constraints will prevent any subgraph solutions from being identified. Constraints are considered satisfied in subgraphs which do not contain any constrained atoms or bonds in either the pattern or target molecules.

The *AddConstraint* method returns `true` if a constraint is added successfully. If the pattern atom or bond in the *OEMatchPair* does not exist as part of the query molecule created in the initialization of the *OESubSearch* object then *AddConstraint* will return `false`. Multiple calls to *AddConstraint* using the same pattern atom or bond will cause previously stored constraints to be overwritten as constraints are mutually exclusive. It is impossible to satisfy multiple simultaneous constraints for a single pattern atom or bond, hence the exclusivity.

A read-only reference to the query molecule *OEQMolBase* contained in an instance of *OEMCSSearch* can be obtained with the *GetPattern* method. Note that if the *OEMCSSearch* was constructed with an *OEQMolBase*, the returned *OEQMolBase* is a separate object. *OEQMolBase* methods can then be used to interrogate the returned *OEQMolBase* reference.

The *SetMaxMatches* method alters the maximum number of maximum common subgraph matches that will be returned by the *OEMCSSearch.Match* method. The search for maximum common substructures will not terminate immediately upon reaching this limit. The maximum common subgraph cannot be known unless the MCS is composed of all atoms and bonds of at least one of the graphs being compared. The limit of subgraphs to be returned may be reached with a smaller subgraph than the maximum. In such a case the search continues for larger subgraphs until the search is exhausted. *OEMCSSearch.Match* will return the first `N` maximum common subgraphs where `N` is less than or equal to the maximum match limit. The default limit set upon construction of an *OEMCSSearch* instance is `1024` matches.

The *SetMinAtoms* method sets the minimum number of atoms required of a subgraph match to be returned by a MCS search. For example, changing the parameter of *SetMinAtoms* in *Listing 5* to 11 would result in no solution since there are only 10 atoms of the largest maximum common substructure (see *Figure: Example of MCSS*).

```
mcss.SetMinAtoms(6);
```

A single atom can be a perfectly valid maximum common subgraph, however, for many applications such a small subgraph may not be considered useful. Setting the minimum number of atoms to a useful size prevents unproductive subgraph matches from being returned by the *OEMCSSearch.Match* method. The default set upon construction of an *OEMCSSearch* instance for the minimum number of atoms is one.

Exhaustive and approximate MCSS

The maximum common substructure search can be performed in two different modes: a very fast method, *OEMCSType.Approximate*, or a more comprehensive method, *OEMCSType.Exhaustive*.

The type of the *OEMCSSearch* can be set at initialization. The default value is *OEMCSType.Exhaustive*.

The approximate method is based on traversing through pre-defined paths of the query structure and trying to map the visited query atoms into target atoms. Because these pre-defined paths represent only a fraction of all possible paths of a compound, it is not guaranteed that the approximate method can find the largest and all common substructures. Significant difference between the detected matches of the two methods could exist in cases when the query or target structure contains complex ring systems (fused or bridged) or stereo centers. However, comparing the two methods for thousands of structures revealed that these cases are rare and the approximate method provides a good trade-off between identifying MCS matches accurately and doing it 3 to 6 times faster than the exhaustive method.

Figure: Approximate MCSS and *Figure: Exhaustive MCSS*, show an example where the substructure identified by the approximate method is smaller by one atom then the solution identified by the exhaustive method.

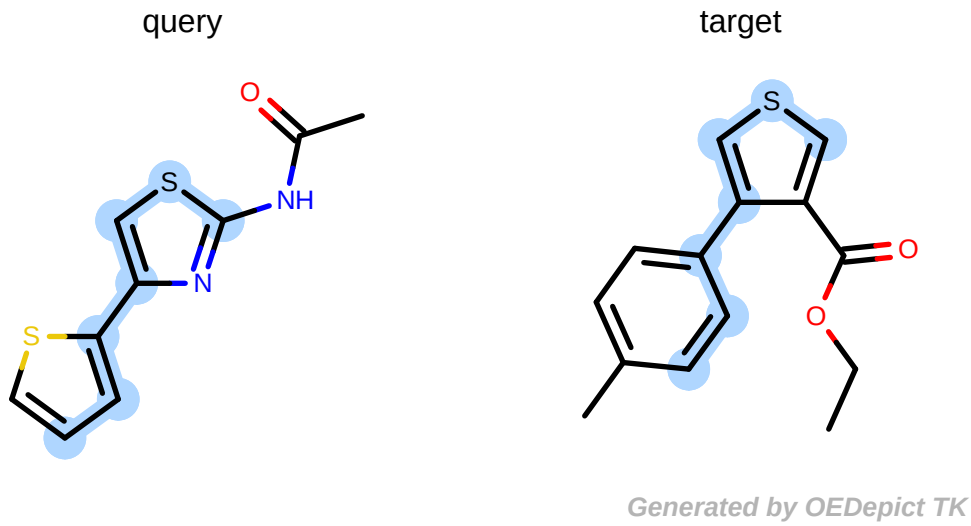


Fig. 18: **Approximate MCSS**
Example for maximum common substructure identified by the approximate method.

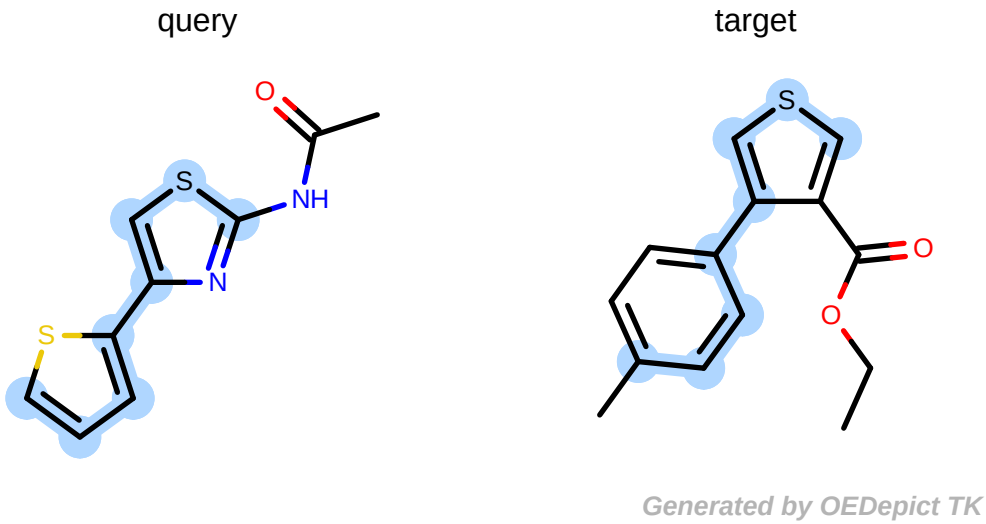


Fig. 19: **Exhaustive MCSS**
Example for maximum common substructure identified by the exhaustive method.

Note

Using the approximate MCS is recommended if the speed of the search is crucial.

MCS scoring functions

OEMCSFunc is an abstract base-class that defines the API used for scoring subgraph matches. The scores generated by implementations of *OEMCSFunc* influence the sorting and retention of maximum common subgraph matches generated by the *OEMCSSearch* class. The scoring function is set by the `OEMCSSearch.SetMCSFunc` method.

It is important to mention that using different scoring functions does not alter the way the search space is traversed to identify common substructures. It affects only how these identified substructures are evaluated.

Four implementations of the *OEMCSFunc* class are available in **OEChem TK**:

OEMCSMaxAtoms

The *OEMCSMaxAtoms* class is designed to order maximum common substructure matches by the maximum number of atoms included in the graph match. If two matches have the same number of atoms, then the tie is split based on the number of bonds contained in the match. (See example in *Scoring A*.)

Scoring function:

$$num.\ of\ mapped\ atoms + \frac{num.\ of\ mapped\ bonds}{100}$$

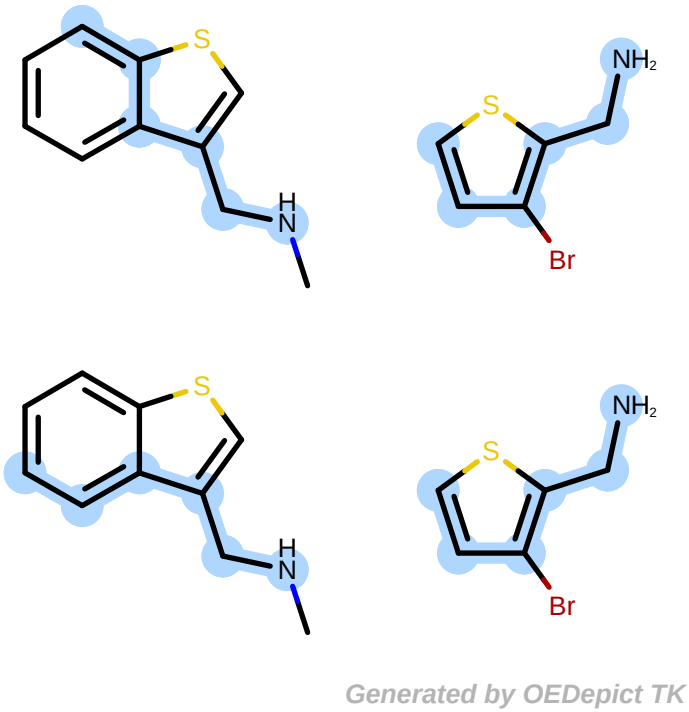


Fig. 20: **Scoring A**
Retrieved matches using 'OEMCSMaxAtoms' as scoring function.

OEMCSMaxBonds

The *OEMCSMaxBonds* class is designed to order maximum common substructure matches by the maximum number of bonds included in the graph match. If two matches have the same number of bonds, then the tie is split based on the number of atoms contained in the match. (See example in *Scoring B*.)

Scoring function:

$$num.\ of\ mapped\ bonds + \frac{num.\ of\ mapped\ atoms}{100}$$

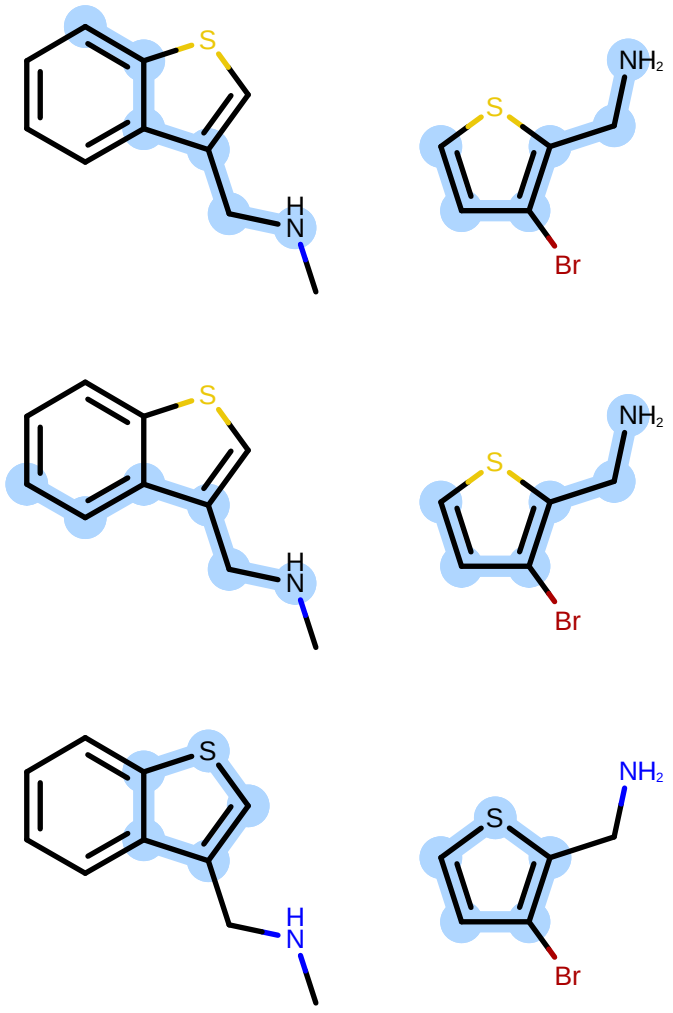


Fig. 21: **Scoring B**
Retrieved matches using 'OEMCSMaxBonds' as scoring function.

OEMCSMaxAtomsCompleteCycles

The *OEMCSMaxAtomsCompleteCycles* class is the same as the *OEMCSMaxAtoms* with the addition of penalizing cyclic query bonds that are not mapped to any target bonds, thereby giving priority to matches which contain complete cycles common to both the pattern and the target structure. (See example in *Scoring C*.)

Scoring function:

$num.\ of\ mapped\ atoms + \frac{num.\ of\ mapped\ bonds}{100} - penalty \times num.\ of\ unmapped\ cyclic\ query\ bonds$

The default penalty for each unmapped cyclic query bond is 1.0.

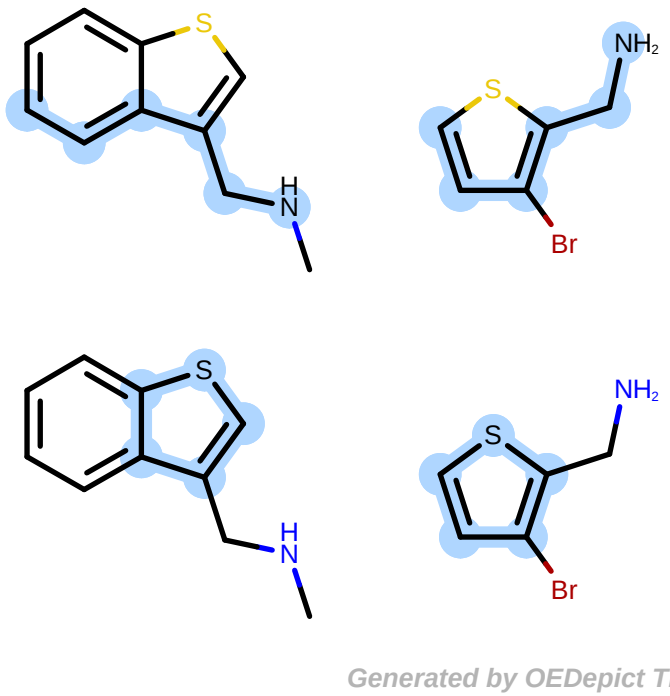


Fig. 22: **Scoring C**
Retrieved matches using 'OEMCSMaxAtomsCompleteCycles' as scoring function.

OEMCSMaxBondsCompleteCycles

The *OEMCSMaxBondsCompleteCycles* class is the same as the *OEMCSMaxBonds* class with the addition of penalizing cyclic query bonds that are not mapped to any target bonds, thereby giving priority to matches which contain complete cycles common to both the pattern and the target structure. (See example in *Scoring D*.)

Scoring function:

$num.\ of\ mapped\ bonds + \frac{num.\ of\ mapped\ atoms}{100} - penalty \times num.\ of\ unmapped\ cyclic\ query\ bonds$

The default penalty for each unmapped cyclic query bond is 1.0.

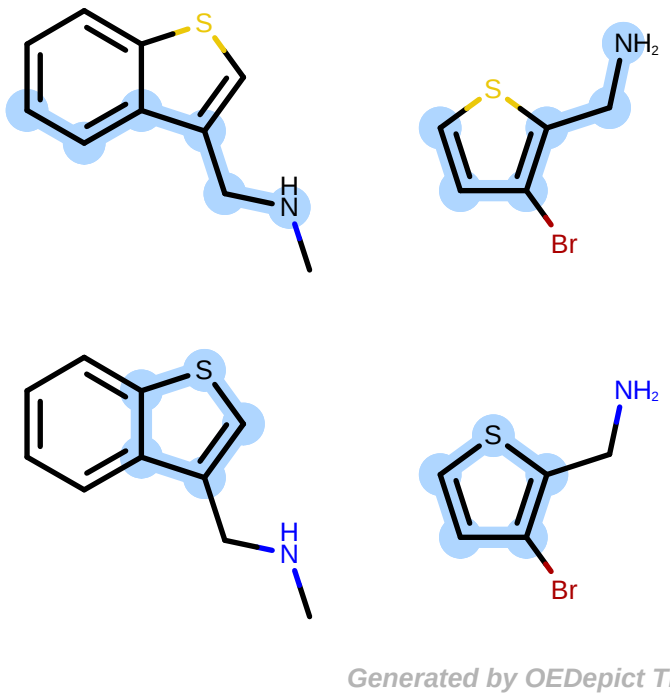


Fig. 23: **Scoring D**
Retrieved matches using 'OEMCSMaxBondsCompleteCycles' as scoring function.

It is important to remember that not only matches with the highest score are retained, but also matches with scores higher than the best score rounded down to the highest integer. In the example shown in *Scoring B* three common substructures are detected using the *OEMCSMaxBonds* scoring function. The first two matches are scored `5.06`, since they are composed of 5 mapped bonds and 6 mapped atoms. There is only one other match which scored higher than 5.0, this is the third retained match with a score of `5.05`.

See also

- [Bunke-2002]
- [McGregor-1982]
- [Raymond-2002]
- [Wang-T-1997]

Clique Search

Clique detection is a bounded common substructure search. It is a useful search method in cases where common substructures other than the maximum common substructure need to be identified. The following example demonstrates a clique search.

Listing 6: Clique search example

```
using System;
using OpenEye.OEChem;

public class CliqueSearch
{
    public static void Main(string[] argv)
    {
        OEGraphMol pattern = new OEGraphMol();
        OEGraphMol target  = new OEGraphMol();
        OEChem.OESmilesToMol(pattern, "c1cc(O)c(O)cc1CCN");
        OEChem.OESmilesToMol(target, "c1c(O)c(O)c(Cl)cc1CCCBr");
        /* create clique search object */
        OECliqueSearch cs = new OECliqueSearch(pattern,
                                               OEExprOpts.DefaultAtoms,
                                               OEExprOpts.DefaultBonds);
        /* ignore cliques that differ by more than 5 atoms from MCS */
        cs.SetSaveRange(5);

        int count = 1;
        /* loop over matches */
        foreach (OEMatchBase match in cs.Match(target))
        {
            Console.WriteLine("Match " + count + " :");

            Console.Write("pattern atoms: ");
            foreach (OEMatchPairAtom ma in match.GetAtoms())
            {
                Console.Write(ma.pattern.GetIdx() + " ");
            }

            Console.WriteLine();

            Console.Write("target atoms: ");
            foreach (OEMatchPairAtom ma in match.GetAtoms())
            {
                Console.Write(" " + ma.target.GetIdx());
```

(continues on next page)

(continued from previous page)

```
            }

            Console.WriteLine();
            count++;
        }
    }
}
```

The same molecules and expression options are used as in *Listing 5*, however, an iterator over all identified cliques is returned by the *OECliqueSearch.Match* method. The *OECliqueSearch.SetSaveRange* method bounds the search. In this case, cliques returned will only differ by five nodes relative to the maximum common substructure. The atom correspondences for each of the returned cliques are printed in the example program.

OEExprOpts Namespace

Pattern matching in **OEChem TK** is always done using query molecules or query graphs. Non-query molecules, *i.e.* those that are defined only by *OEMolBase* abstract base-class must be converted into a query molecule. Conversion into a query molecule is controlled using the values in the *OEExprOpts* namespace. Expression options can either be specified in the constructor for an *OEQMol*, or using the convenience constructors in the pattern matching classes, *OESubSearch*, *OEMCSSearch*, *OECliqueSearch* which take expression options as arguments.

Figure: Default example shows an example where the maximum common substructure search is performed using the *OEExprOpts.DefaultAtoms* and *OEExprOpts.DefaultBonds* options.

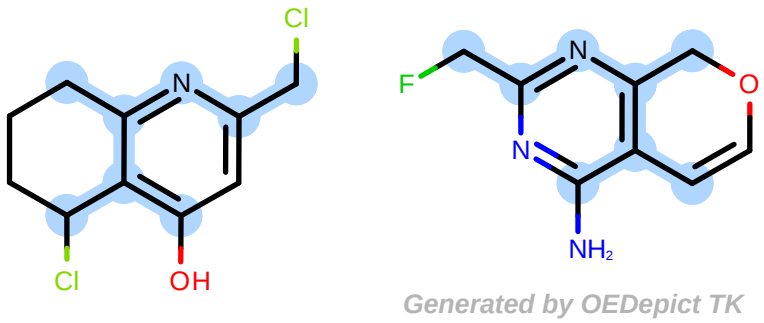


Fig. 24: **Default example**
Example of maximum common substructure search with 'DefaultAtoms' and 'DefaultBonds' options

The *OEExprOpts.DefaultAtoms* option means that two atoms are considered to be equivalent if they have the same atomic number, aromaticity, and formal charge. The *OEExprOpts.DefaultBonds* option means that two bonds can be mapped to each other if they have the same bond order and aromaticity.

Listing 7: MCSS with atom and bond expressions

```
using System;
using OpenEye.OEChem;

public class MCSSOEExprOpts
{
    public static int Main(string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        OEGraphMol pattern = new OEGraphMol();
        OEGraphMol target  = new OEGraphMol();
        OEChem.OESmilesToMol(pattern, "c1(cc(nc2c1C(CCC2)Cl)CCl)O");
        OEChem.OESmilesToMol(target,  "c1(c2c(nc(n1)CF)COC=C2)N");

        uint atomexpr = OEExprOpts.DefaultAtoms;
        uint bondexpr = OEExprOpts.DefaultBonds;

        OEQMol patternQ = new OEQMol(pattern);
        /* generate query with atom and bond expression options */
        patternQ.BuildExpressions(atomexpr, bondexpr);
        OEMCSSearch mcss = new OEMCSSearch(patternQ.QMol());

        bool unique = true;
        int count = 1;
        /* loop over matches */
        foreach (OEMatchBase match in mcss.Match(target, unique))
        {
            Console.WriteLine("Match " + count + ":");
            Console.WriteLine("Number of matched atoms: " + match.NumAtoms());
            Console.WriteLine("Number of matched bonds: " + match.NumBonds());
            /* create match subgraph */
            OEGraphMol m = new OEGraphMol();
            OEChem.OESubsetMol(m, match, true);
            Console.WriteLine("match smiles = " + OEChem.OEMolToSmiles(m));
            count++;
        }
        return 0;
    }
}
```

The best way to understand how various atom and bond expressions influence the pattern matching is to change the atom and bond expressions in *Listing 7* and compare the obtained matches.

```
uint atomexpr = OEExprOpts.DefaultAtoms;
uint bondexpr = OEExprOpts.DefaultBonds;
```

After constructing the pattern molecule, the *BuildExpressions* method defines the level of atom and bond matching between the pattern molecule and any target molecule.

```
patternQ.BuildExpressions(atomexpr, bondexpr);
```

By modifying the atom and bond expression options, very diverse pattern matching can be performed. *Figure: Example A* – *Figure: Example E* show several examples where maximum common substructure searches are performed for the same query and target molecules, but with various atom and bond expression options.

In the first example in *Figure: Example A*, the *OEExprOpts.ExactAtoms* expression option is used to give a higher degree of discrimination of the equivalence of atoms, *i.e.* atoms can only be mapped to each other if they have the same degree, number of hydrogens, chirality, mass, and ring membership, in addition to the requirements of the *OEExprOpts.DefaultAtoms* option.

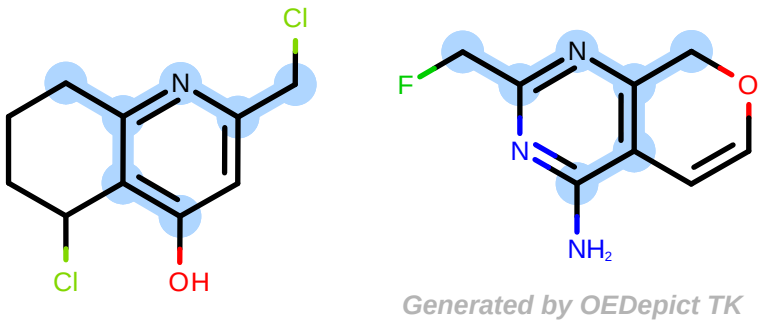


Fig. 25: **Example A**
`ExactAtoms` and 'DefaultBonds`

Figure: Example B – *Figure: Example E* show examples where the discrimination capability of the `OEExprOpts.DefaultAtoms` is decreased by adding various modifiers. For example, using the `OEExprOpts.EqAromatic` modifier, atoms in any aromatic ring systems are considered equivalent. As a result, the pyridine and pyrimidine ring can be mapped to each other in *Figure: Example B*. Similarly, `OEExprOpts.EqHalogen` (*Figure: Example C*) and `OEExprOpts.EqONS` (*Figure: Example D*) define equivalency between halogen atoms and oxygen-nitrogen-sulfur atoms, respectively. Using `OEExprOpts.EqCAliphaticONS` (*Figure: Example E*) an aliphatic query carbon atom is considered equivalent to any oxygen, nitrogen, or sulfur atom.

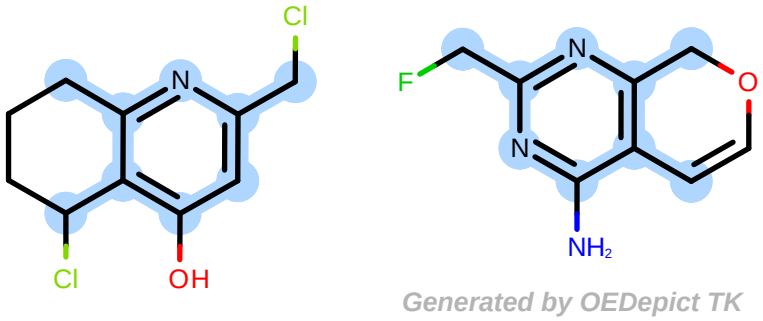


Fig. 26: **Example B**
'DefaultAtoms|EqAromatic' and 'DefaultBonds'

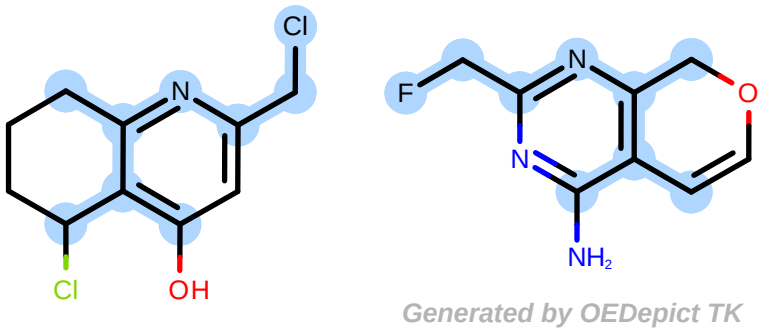


Fig. 27: **Example C**
'DefaultAtoms|EqHalogen' and 'DefaultBonds'

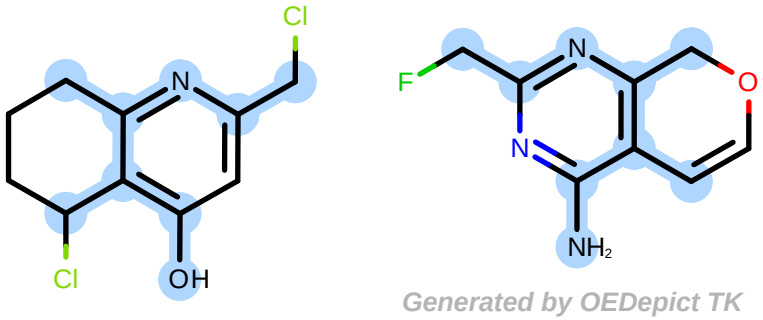


Fig. 28: **Example D**
'DefaultAtoms|EqONS' and 'DefaultBonds'

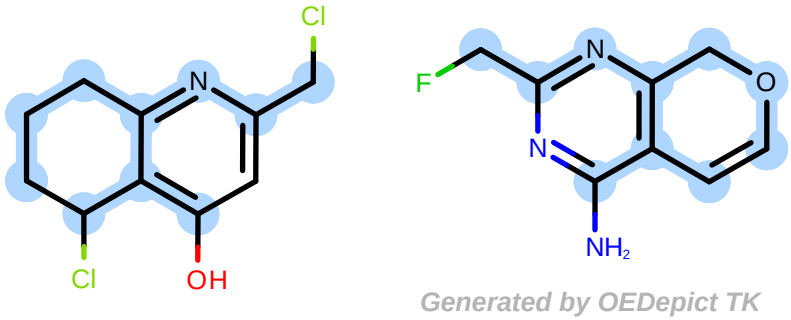


Fig. 29: **Example E**
'DefaultAtoms|EqCAliphaticONS' and 'DefaultBonds'

Similar modifiers exist for altering bond equivalency. *Figure: Example F* shows an example where single and double bonds are considered identical when `OEExprOpts.EqSingleDouble` modifier is utilized.

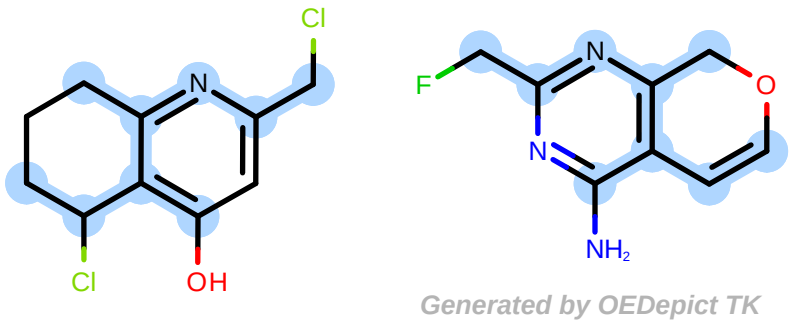


Fig. 30: **Example F**
'DefaultAtoms' and 'DefaultBonds|EqSingleDouble'

The last example in *Figure: Example G* represents a very unrestrained search, where both the atom and bond expression options have weak discrimination power.

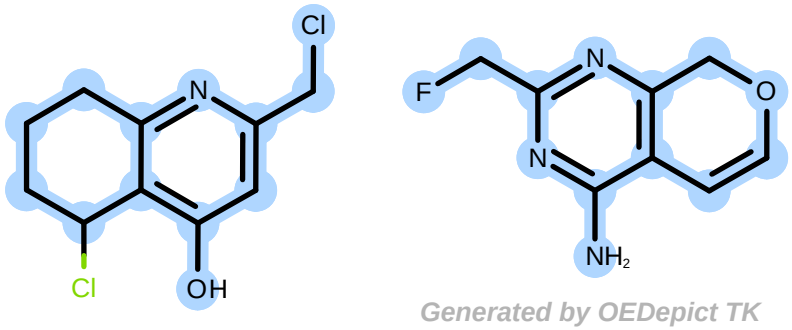


Fig. 31: **Example G**
'DefaultAtoms|EqAromatic|EqCAliphaticONS|EqHalogen|EqONS' and 'DefaultBonds|EqSingleDouble'

Even though only maximum common substructure search examples are presented here, atom and bond expression options can be similarly used with substructure searches or clique detections. For a full description of expression options and their usage please refer to the `OEExprOpts` namespace section in **OEChem TK** API.

2.1.18 Substructure Search with MDL Queries

Section *Substructure Search* describes how to perform substructure search initialized with a SMILES or SMARTS string. **OEChem TK** also provides the ability to interpret and utilize query structures expressed in the MDL query file format (see MDL query example in *Figure: Example of MDL query substructure*). `Listing 1` shows how to initialize an `OESubSearch` object from a MDL query file and perform a substructure search.

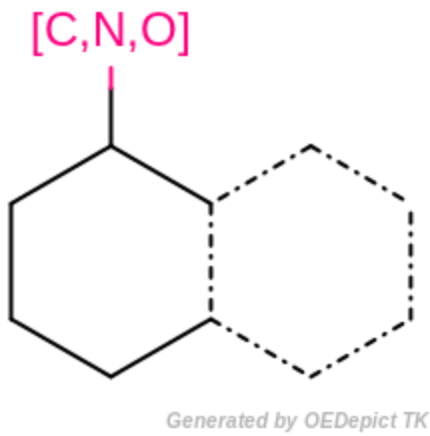


Fig. 32: **Example of MDL query substructure**

Listing 1: Example of substructure search using MDL query file

```
using System;
using OpenEye.OEChem;

public class MDLQuerySearchSingleMatch
{
    public static int Main(string[] args)
    {
        oemolistream qfile = new oemolistream("query.mol");
        oemolistream tfile = new oemolistream("targets.sdf");

        // set the same aromaticity model for the query and the target file
        uint aromodel = OEIFlavor.Generic.OEAroModelMDL;
        uint qflavor = qfile.GetFlavor(qfile.GetFormat());
        qfile.SetFlavor(qfile.GetFormat(), (qflavor | aromodel));
        uint tflavor = tfile.GetFlavor(tfile.GetFormat());
        tfile.SetFlavor(tfile.GetFormat(), (tflavor | aromodel));

        // read MDL query and initialize the substructure search
        uint opts = OEMDLQueryOpts.Default | OEMDLQueryOpts.SuppressExplicitH;
        OEQMol qmol = new OEQMol();
        OEChem.OEReadMDLQueryFile(qfile, qmol, opts);
        OESubSearch ss = new OESubSearch(qmol);
        // loop over target structures
        int tindex = 1;
        OEGraphMol tmol = new OEGraphMol();

        while (OEChem.OEReadMolecule(tfile, tmol))
        {
            OEChem.OEPrepareSearch(tmol, ss);
            if (ss.SingleMatch(tmol))
            {
```

(continues on next page)

(continued from previous page)

```
                Console.WriteLine("hit target = " + tindex);
            }

            tindex += 1;
        }
        return 0;
    }
}
```

After opening the MDL query and the target files, the model used to assign aromaticity to the imported structures can be adjusted.

```
uint aromodel = OEIFlavor.Generic.OEAroModelMDL;
uint qflavor = qfile.GetFlavor(qfile.GetFormat());
qfile.SetFlavor(qfile.GetFormat(), (qflavor | aromodel));
uint tflavor = tfile.GetFlavor(tfile.GetFormat());
tfile.SetFlavor(tfile.GetFormat(), (tflavor | aromodel));
```

If the aromaticity model is not specified for the input files, then the **OpenEye**'s aromaticity model is used by default. For more information about the various aromaticity models of **OEChem TK** see *Aromaticity Perception*.

```
OEChem.OEReadMDLQueryFile(qfile, qmol, opts);
OESubSearch ss = new OESubSearch(qmol);
```

In general, the aromaticity model chosen should be consistent between the query and target molecules to be searched. Using different aromaticity models may produce false negatives as aromatic systems may be treated differently. Section *Aromaticity* further explains the effects of using various aromaticity models when performing a substructure search.

OEReadMDLQueryFile function reads the MDL query directly into a *OEQMolBase* object , which then can be used to initialize an *OESubSearch* instance.

The MDL query structure can also be read into a *OEMolBase* object (see code snippet below). In this case, the *OEReadMDLQueryFile* function attaches the query features present in the input MDL file to the related atoms and bonds of the *OEMolBase* object. The *OEQMolBase* object can be subsequently created by calling the *OEBuildMDLQueryExpressions* function.

```
OEGraphMol mol = new OEGraphMol();
OEChem.OEReadMDLQueryFile(qfile,mol);
// mol can be manipulated here
OEQMol qmol = new OEQMol();
// build OEQMol with OEMDLQueryOpts.Default option
OEChem.OEBuildMDLQueryExpressions(qmol,mol);
OESubSearch ss = new OESubSearch(qmol);
```

The declaration of these functions are:

```
bool OEReadMDLQueryFile(oemolistream ifs, OEMolBase mol);
bool OEReadMDLQueryFile(oemolistream ifs, OEQMolBase qmol, int opts);
// opts (default value = OEMDLQueryOpts.Default)
bool OEBuildMDLQueryExpressions(OEQMolBase qmol, OEMolBase mol, uint opts);
// opts (default value = OEMDLQueryOpts.Default)
```

MDL Query Interpretation

The *opts* parameter defines how the MDL query is interpreted when an `OEQMolBase` object is constructed. The following options are present in the `OEMDLQueryOpts` namespace:

1. `OEMDLQueryOpts.Default`

   Only constraints explicitly specified in the MDL file are added to the *OEQMolBase* query structure. See section *Supported MDL Query Features* for the supported MDL query features.

2. `OEMDLQueryOpts.SuppressExplicitH`

   This option controls how the explicit hydrogens of the query are matched to the explicit/implicit hydrogens of the target structures. For more information see *Explicit Hydrogens*.

3. `OEMDLQueryOpts.AddBondAliphaticConstraint`

   If this option is specified, then an aliphatic query bond can only be mapped to the aliphatic bonds in the target structure. *Figure: Interpretation A* shows how the MDL query structure is interpreted when the `OEMDLQueryOpts.AddBondAliphaticConstraint` option is used.

   Query 'A' will match all three of the target compounds displayed in *Figure: Interpretation A*. If the *OpenEye* model is used to perceive aromatic rings, then query 'B' substructure is present only in target '2'. If the MDL aromaticity model is used, then target '3' is also a hit, since five-membered heterocycles are not considered aromatic in this model. For more information about different aromaticity models and their effects on substructure searching see section *Aromaticity*.

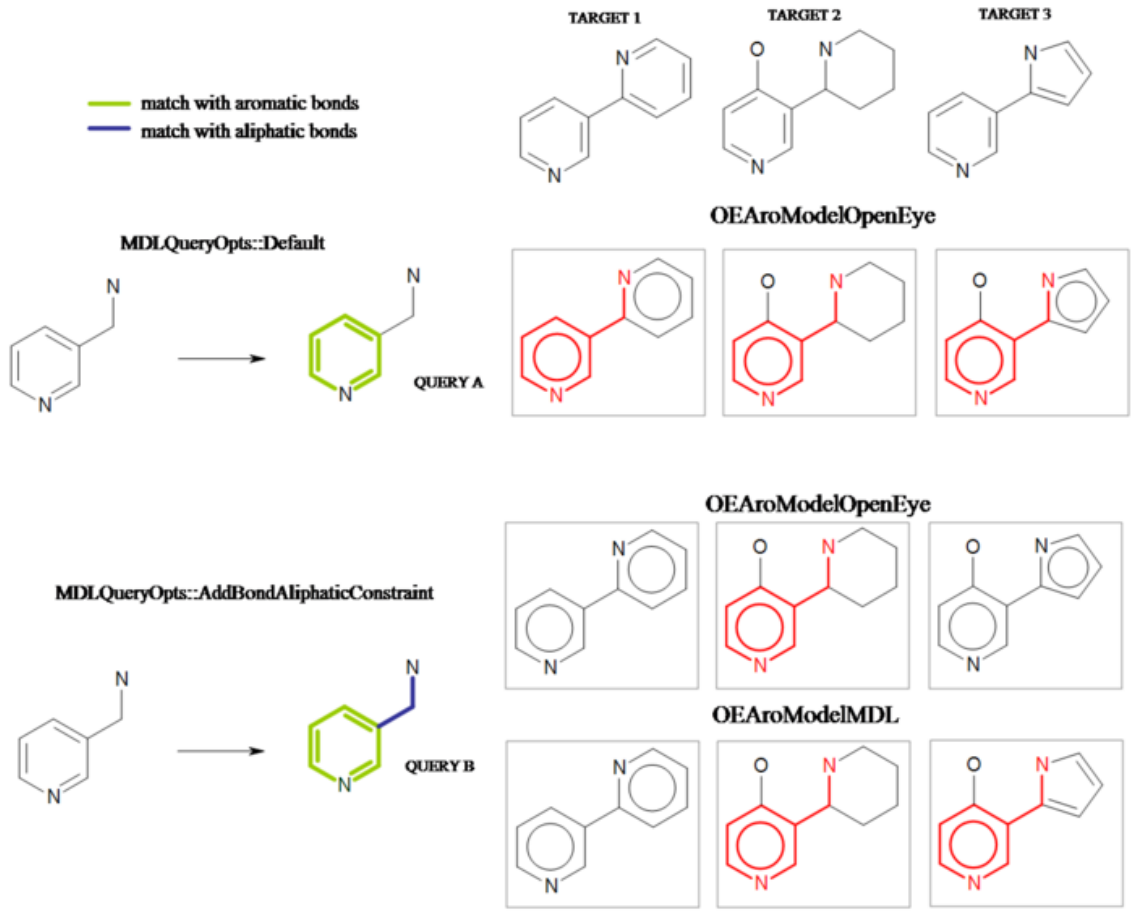


Fig. 33: **Interpretation A**
Interpretation of the MDL query with 'AddBondAliphaticConstraint' option.

4. `OEMDLQueryOpts.AddBondTopologyConstraint`

   By default, ring bonds in the query structure can only be mapped to ring bonds in the target, and chain bonds in the query can be mapped to both ring and chain bonds. The `OEMDLQueryOpts.AddBondTopologyConstraint` option enables enforcement of bond topology constraint during substructure search i.e. ring bonds in the query can only be mapped to ring bonds, and chain bonds in the query can only be mapped to chain bonds.

   Figure: Interpretation B shows how the MDL query structure is interpreted when the `OEMDLQueryOpts.AddBondTopologyConstraint` option is used. Query 'A' will match all three of the target compounds displayed in *Figure: Interpretation B*, while query 'B' is present only in target '3'.

   If the 'ring' property is specified in the MDL query file for a particular chain bond in the query structure, then its topology constraint is overridden.

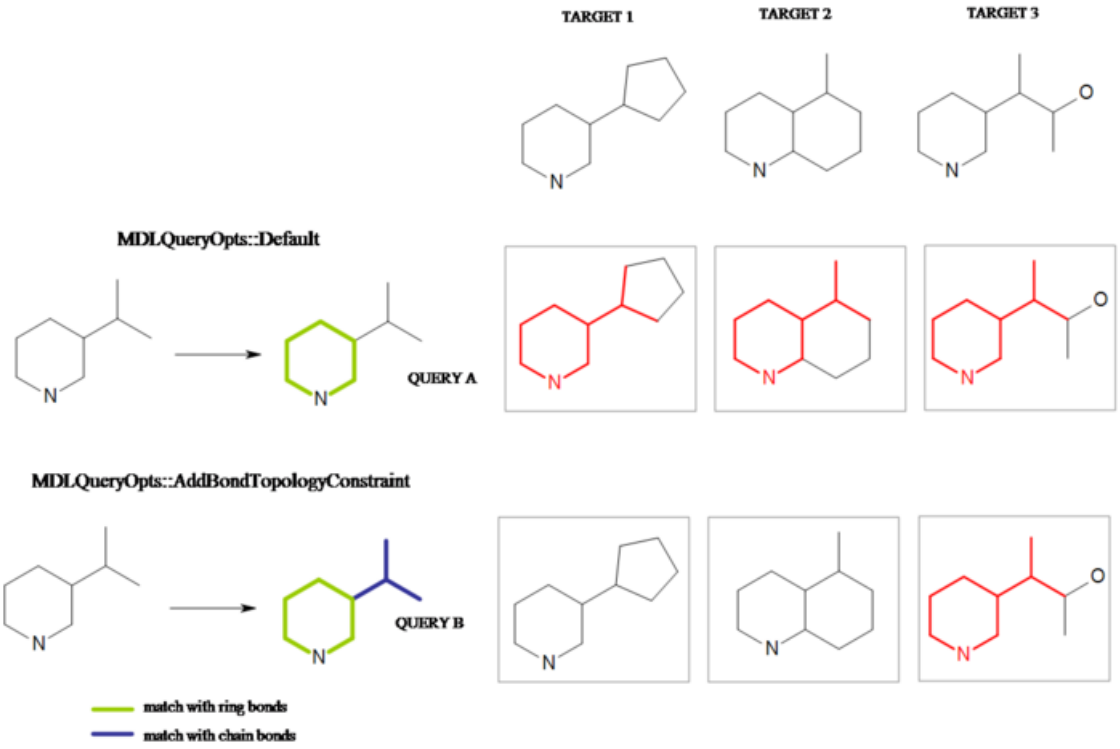


Fig. 34: **Interpretation B**
Interpretation of the MDL query with `AddBondTopologyConstrain' option.

5. *OEMDLQueryOpts.MatchAtomStereo*

   When this option is enabled, an S/R atom stereo configuration in the query will only match any S/R configuration in the target molecule, but not to a R/S (opposite) or an unspecified one. If an atom stereo configuration is undefined in the query, it is allowed to match to any target atom regardless of its stereo configuration.

6. *OEMDLQueryOpts.MatchIsotope*

   When this option is enabled, query atoms with an isotope can match to a target atom only if they have the same atomic mass. If the query atom has no specified isotope number, it will match to any target atom regardless of its atomic mass.

Supported MDL Query Features

The table below summarizes the MDL query features currently supported by the **OEChem TK**. These query feature can only be read with the low level *OEReadMDLQueryFile* function and not the high-level *OEReadMolecule* function.

Table 12: **Summary of supported MDL query features in V2000 and V3000**

CTFile V2K	CTFile V3K	Notes
`M CHG`	`CHG=`	formal charge
`<atomblock>`	`HCOUNT=`	number of allowed hydrogens V2K:0=off, 1=H0, 2-5 for H1-H4; V3K:0=off, -1=H0, 1-5 for H1-H5
`M ISO`	`MASS=`	mass difference
`M RBC`	`RBCNT=`	limit of number of allowed ring bonds attached to an atom
`M SUB`	`SUBST=`	number of allowed substitutions of an atom
`M ALS`	`[aaa,bbb,..]`	list alternative atom types for an atom
`<atomblock>`	`STBOX=`	stereo care (e.g. for c/t doubles)
`L,A,Q`	`L,A,Q`	L=atom list, A=Any atom except hydrogen, Q=Any atom except hydrogen and carbon
`M UNS`	`UNSAT=`	specifies whether or not an atom is unsaturated, *i.e.*, having at least one multiple bond
`<bondblock>`	`STBOX=`	bond stereochemistry that is considered, if both ends of the bond are marked with *stereo care* flags in the atom block
`<bondblock>`	`TOPO=`	bond topology (0=off, 1=ring, 2=chain)
`<bondblock>`	`<bondblock>`	4-aromatic, 5=single or double, 6=single or aromatic, 7=double or aromatic, 8=any

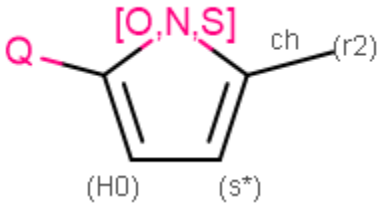


Fig. 35: **Query**
Example of MDL query structure.

See also

- MDL Query Depiction chapter for the depiction of the supported query features
- Accelrys CTfile Formats document for more information about the atom and bond query features

Aromaticity

Perceiving aromaticity in the query and the target structures is important in order to ensure that the result of a substructure search is independent of the different Kekulé representations of the participating structures. A query bond which is part of an aromatic ring system can be mapped to any aromatic bonds in the target. *Figure: Aromaticity Match* shows an example where both Kekulé representations of the benzene-1,2-diol substructure are present in the two target structures.

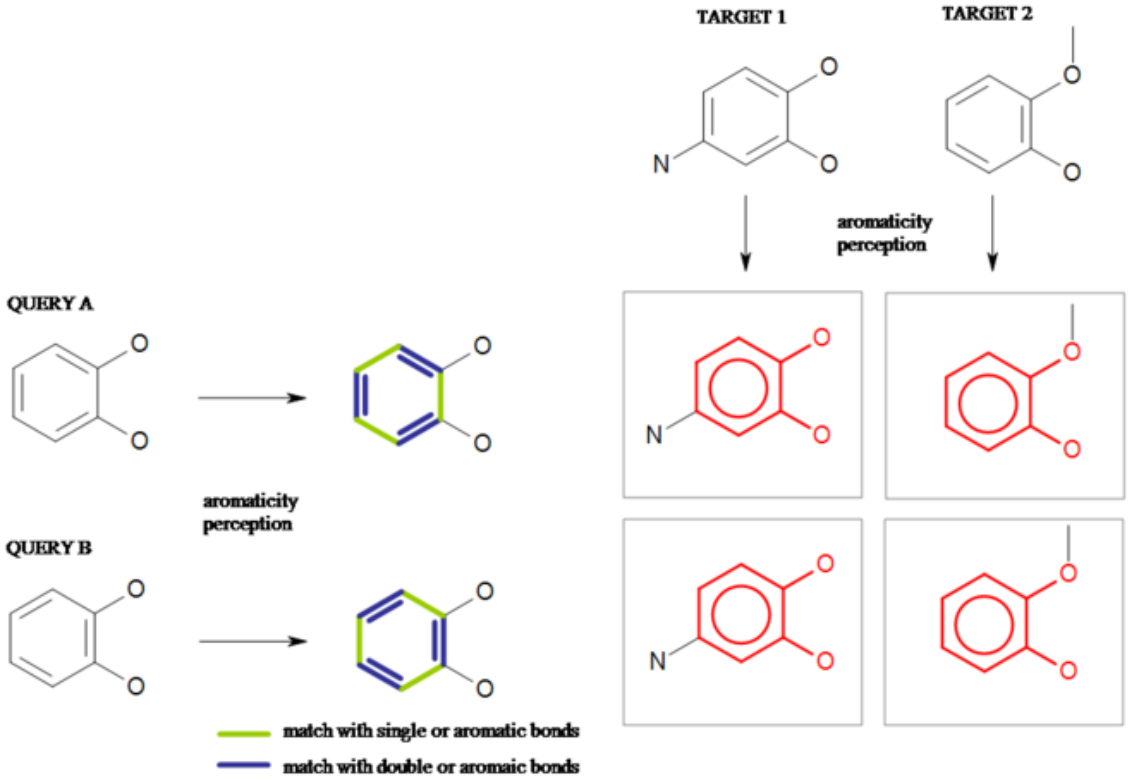


Fig. 36: **Aromaticity Match**
The result of substructure search is independent of the Kekulé representations of the participating structures.

Altering the aromaticity model will affect the results of a substructure search. *Figure: Aromaticity A* – *Figure: Aromaticity C* show several examples where different results were obtained by applying the MDL and the OpenEye aromaticity models. This is a consequence of the fact that in the MDL aromaticity model, five-membered heterocycles are not considered aromatic.

Note

It is highly recommended to apply the same aromaticity model to the query and the target structures.

Listing 1 shows an example of how to change the aromaticity flavor of input files.

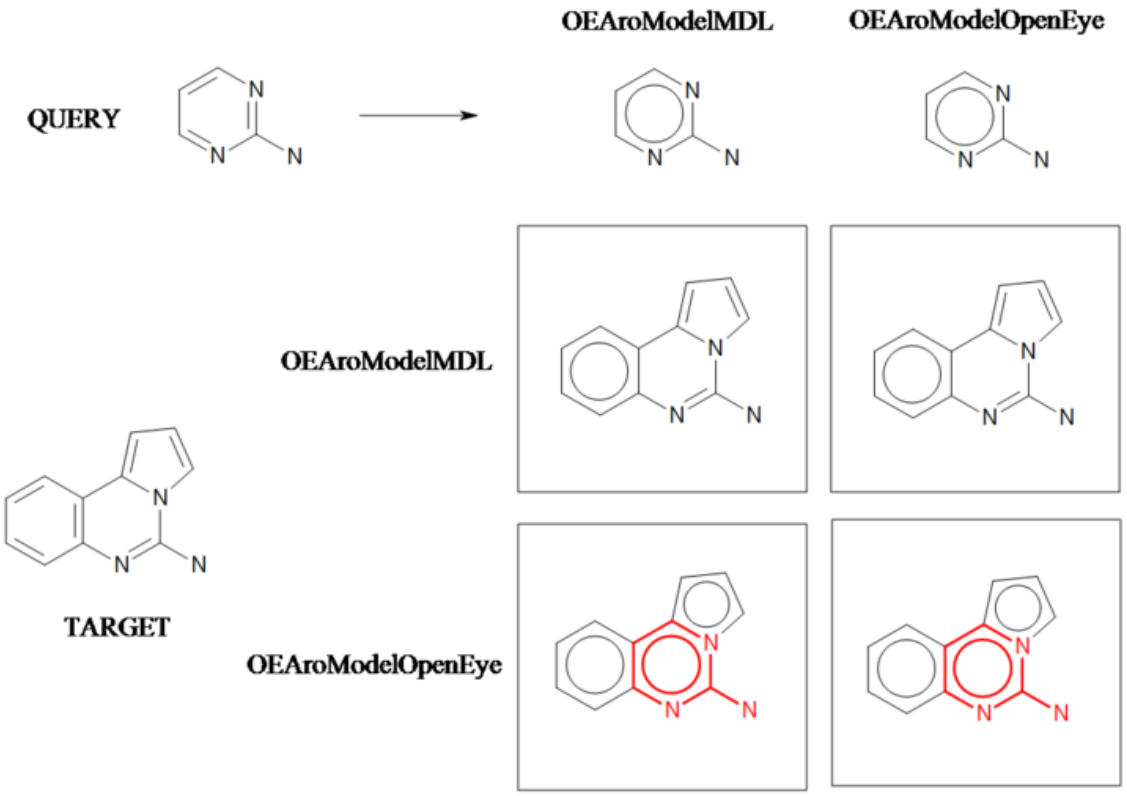


Fig. 37: **Aromaticity A**
Example of a substructure search applying different aromaticity models.

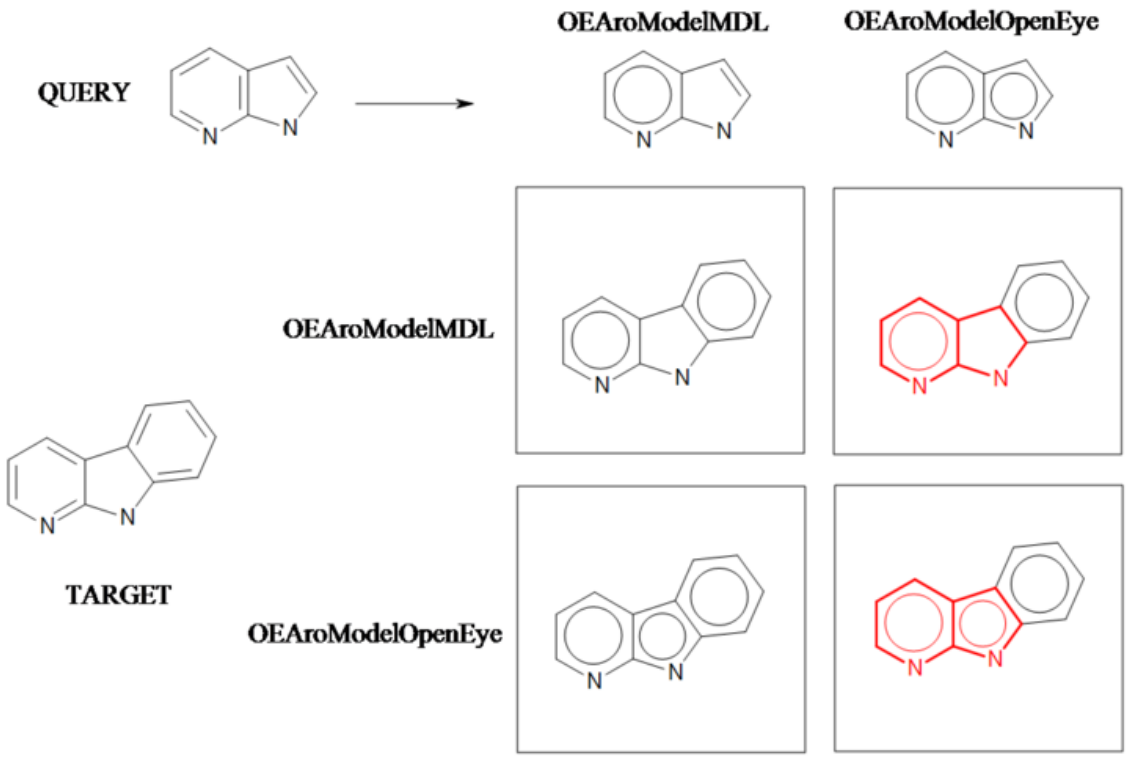


Fig. 38: **Aromaticity B**
Example of a substructure search applying different aromaticity models.

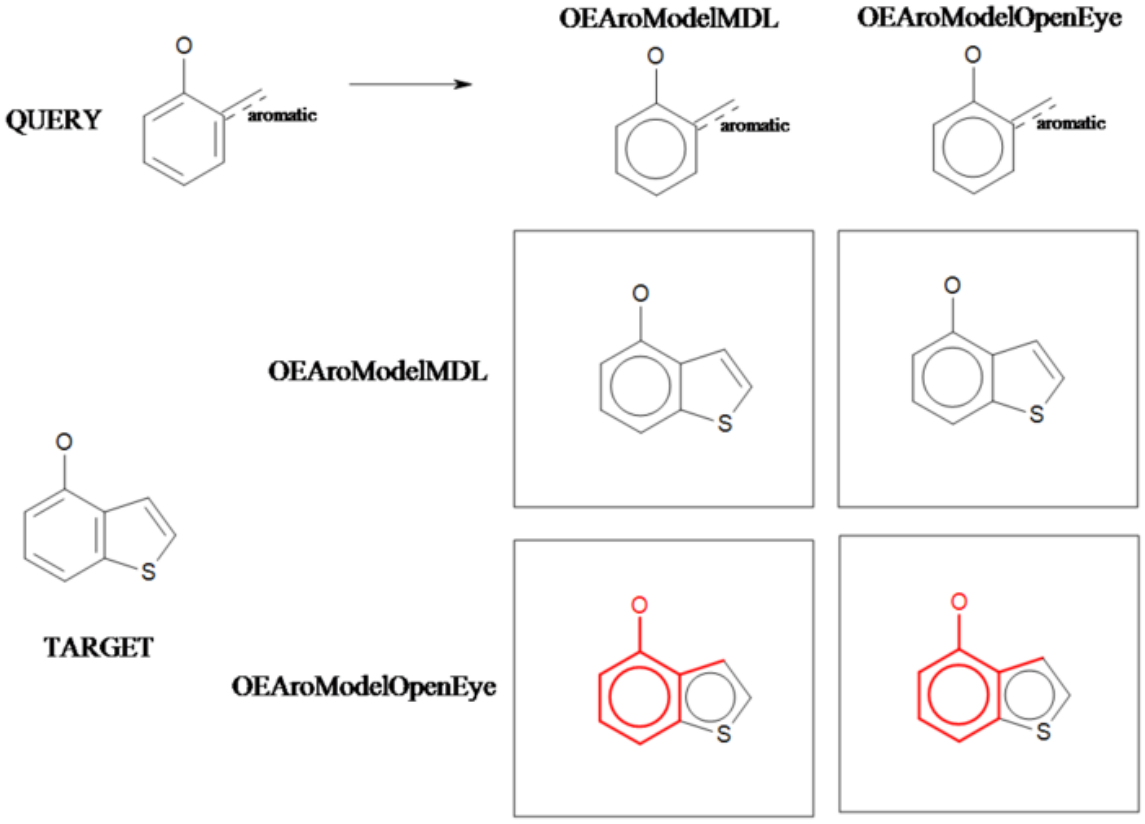


Fig. 39: **Aromaticity C**
Example of a substructure search applying different aromaticity models.

Aromaticity with Generic Atoms

If a ring in the query structure contains generic atom(s) (see example in *Figure: Generic atom example A*), then the aromaticity of the ring can not be perceived. In order to maintain the independence from the Kekulé representation, 6-membered rings with alternating single/double bonds are assumed to be aromatic.

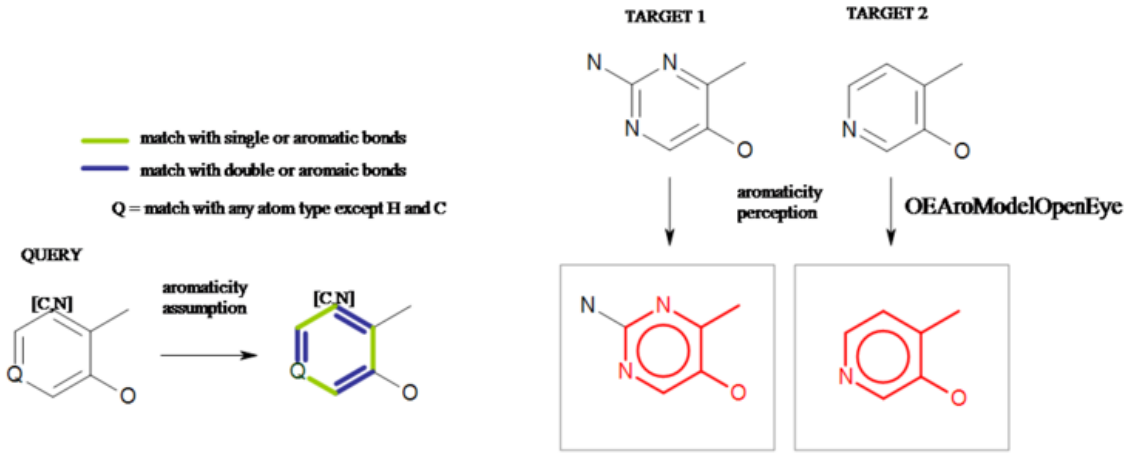


Fig. 40: **Generic atom example A**
Example of aromaticity assumption in a 6-membered ring with generic atoms.

Similarly, a 5-membered ring with generic atom(s) is considered aromatic if it is composed of two single and two double bonds. See Example in *Figure: Generic atom example B*.

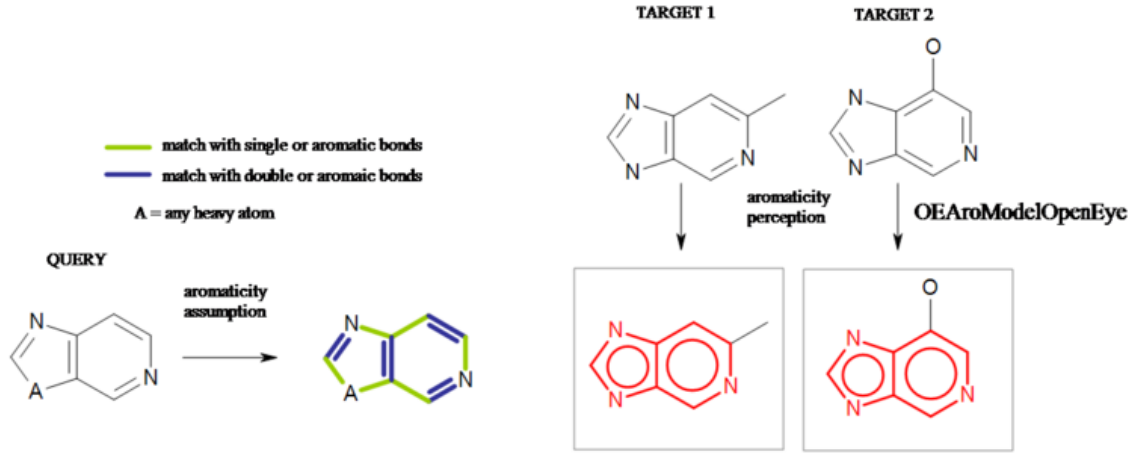


Fig. 41: **Generic atom example B**
Example of aromaticity assumption in a 5-membered ring with generic atoms.

Explicit Hydrogens

During the substructure search, each query atom has to be mapped to a target atom in order to detect subgraph isomorphism. Therefore, a problem can arise if the query structure contains explicit hydrogens or an atom list with hydrogens (see example in *Figure: Query*), but the target structure has implicit hydrogens *Figure: Targets*)

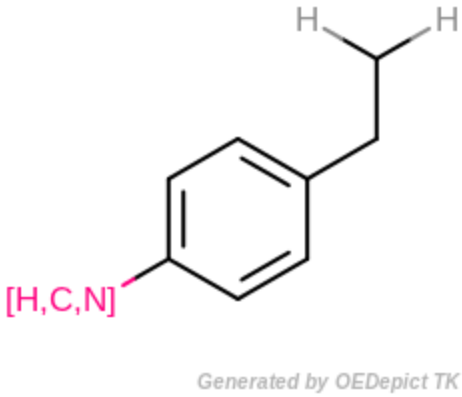


Fig. 42: **Example of MDL query structure with explicit hydrogens**

Listing 2: Example of substructure search with accessing atom mapping

```
using System;
using OpenEye.OEChem;

public class MDLQuerySearchMapping
{
    public static int Main(string[] args)
    {
        oemolistream qfile = new oemolistream("query.mol");
        oemolistream tfile = new oemolistream("targets.sdf");

        //read MDL query and initialize the substructure search
        OEQMol qmol = new OEQMol();
        OEChem.OEReadMDLQueryFile(qfile, qmol);
        OESubSearch ss = new OESubSearch(qmol);

        // loop over target structures
        int tindex = 1;
        OEGraphMol tmol = new OEGraphMol();
        while (OEChem.OEReadMolecule(tfile, tmol))
        {
            OEChem.OEAddExplicitHydrogens(tmol);
            OEChem.OEPrepareSearch(tmol, ss);
            foreach (OEMatchBase match in ss.Match(tmol, true))
            {
                Console.Write("hit target = " + tindex);

                foreach (OEAtomBase a in match.GetTargetAtoms())
                {
                    Console.Write(" " + a.GetIdx() + OEChem.OEGetAtomicSymbol(a.
↪GetAtomicNum()));
                }
```

(continues on next page)

(continued from previous page)

```
                Console.WriteLine();
            }

            tindex += 1;
        }
        return 0;
    }
}
```

This problem can be solved in two ways:

1. The explicit hydrogens in the query molecule can be suppressed during *OEQMolBase* construction by using the *OEMDLQueryOpts.SuppressExplicitH* option (see example in *Listing 1*). A query atom can only be mapped to a target atom if it has at least as many implicit hydrogens as explicit hydrogens were removed from the query atom. This solution is recommended only if the presence or absence of the query substructure is of interest, but not the complete mapping between the query and the target. The *SingleMatch* function returns whether or not the query is present in the target, but the mapping is not accessible.
2. If the complete mapping between the query and the target is of interest, the *OEAddExplicitHydrogens* function has to be called for each target structure before performing the substructure search. (See example in *Listing 2*).

In both cases, the query presented in *Figure: Query* will match only target 'C' and 'D' shown in *Figure: Targets*. *Figure: Atom mapping* shows the three detected substructures in target 'C' when adding explicit hydrogens to the target structures.

The execution of the substructure search is significantly faster if the hydrogens are suppressed in the target structures, since the search space can be an order of magnitude smaller.

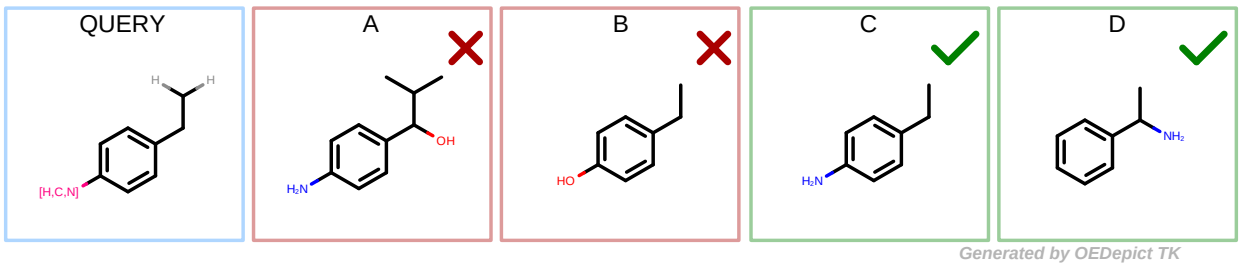


Fig. 43: **Targets – Example of target structures with implicit hydrogens**

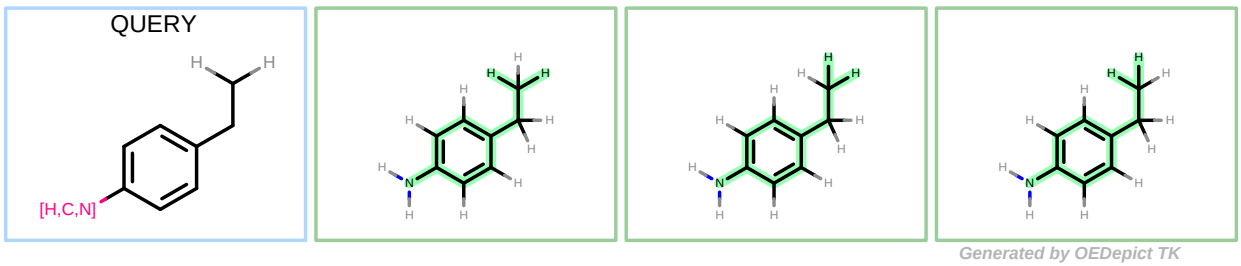


Fig. 44: **Atom mapping – Example of substructure search with atom mapping**

2.1.19 Reactions

Reaction processing in **OEChem TK** is divided into two categories: unimolecular reactions and library generation. Unimolecular reactions are useful for (although not limited to) normalization reactions. The `OEUniMolecularRxn` class in **OEChem TK** applies chemical transformations to individual molecules. Reactions can also be used to generate combinatorial libraries using **OEChem TK**. Both 'clipping' and reaction based enumeration can be achieved using the `OELibraryGen` class.

Figure: OEUniMolecularRxn and *Figure: OELibraryGen* demonstrate the difference between the operation of `OEUniMolecularRxn` and the operation of `OELibraryGen`.

`OEUniMolecularRxn` always takes one starting compound (see *Figure: OEUniMolecularRxn*) to which the specified transformation is applied recursively, until no further reactant pattern are matched, thereby generating one final product. See more details about `OEUniMolecularRxn` class in section *Normalization Reactions*.

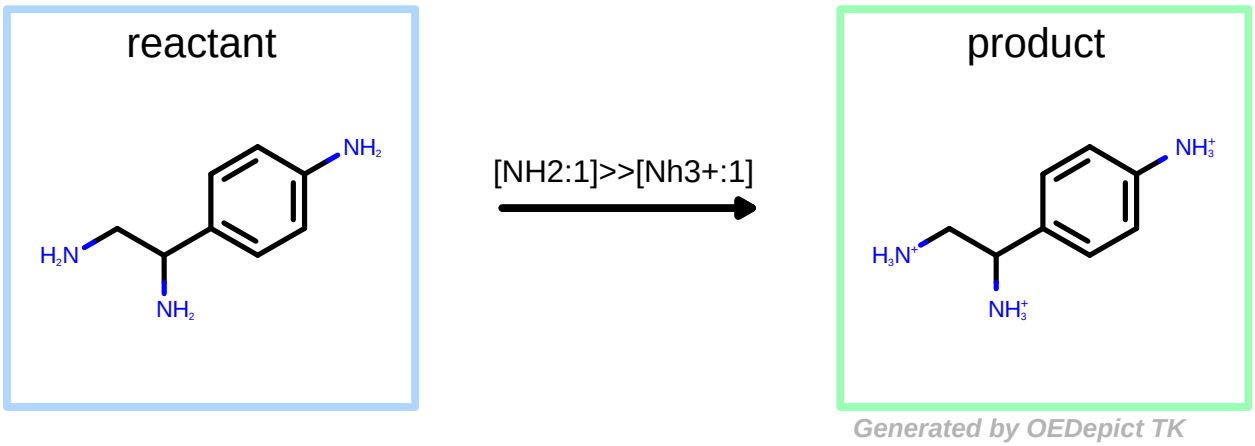


Fig. 45: **OEUniMolecularRxn**
Example of amine nitrogen protonation using 'OEUniMolecularRxn'. Transformations are applied to the starting material recursively

`OELibraryGen` performs transformations independently to each other, enumerating all possible combinations. As a result, three different products are generated when the same protonation transformation is applied to the starting material using `OELibraryGen` (see *Figure: OELibraryGen*).

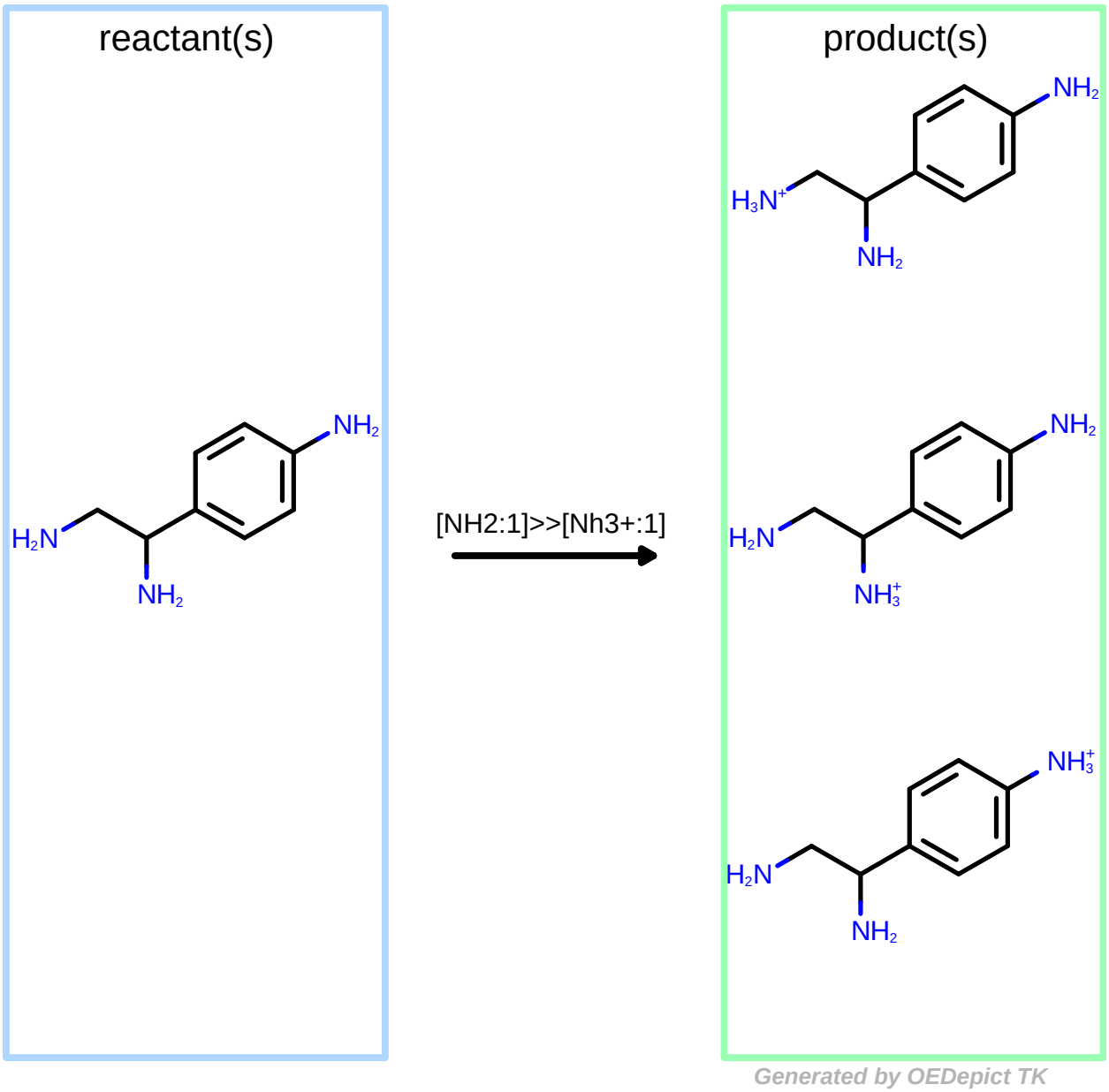


Fig. 46: **OELibraryGen**
Example of amine nitrogen protonation using 'OELibraryGen'. Transformations are applied to the starting material independently

When `OELibraryGen` is used to generate combinatorial libraries (see *Figure: Library generation*), all starting material combinations are enumerated to generate products by considering that any reactant patterns can be mapped to any starting materials multiple times. See more details about `OELibraryGen` class in section *Library Generation*.

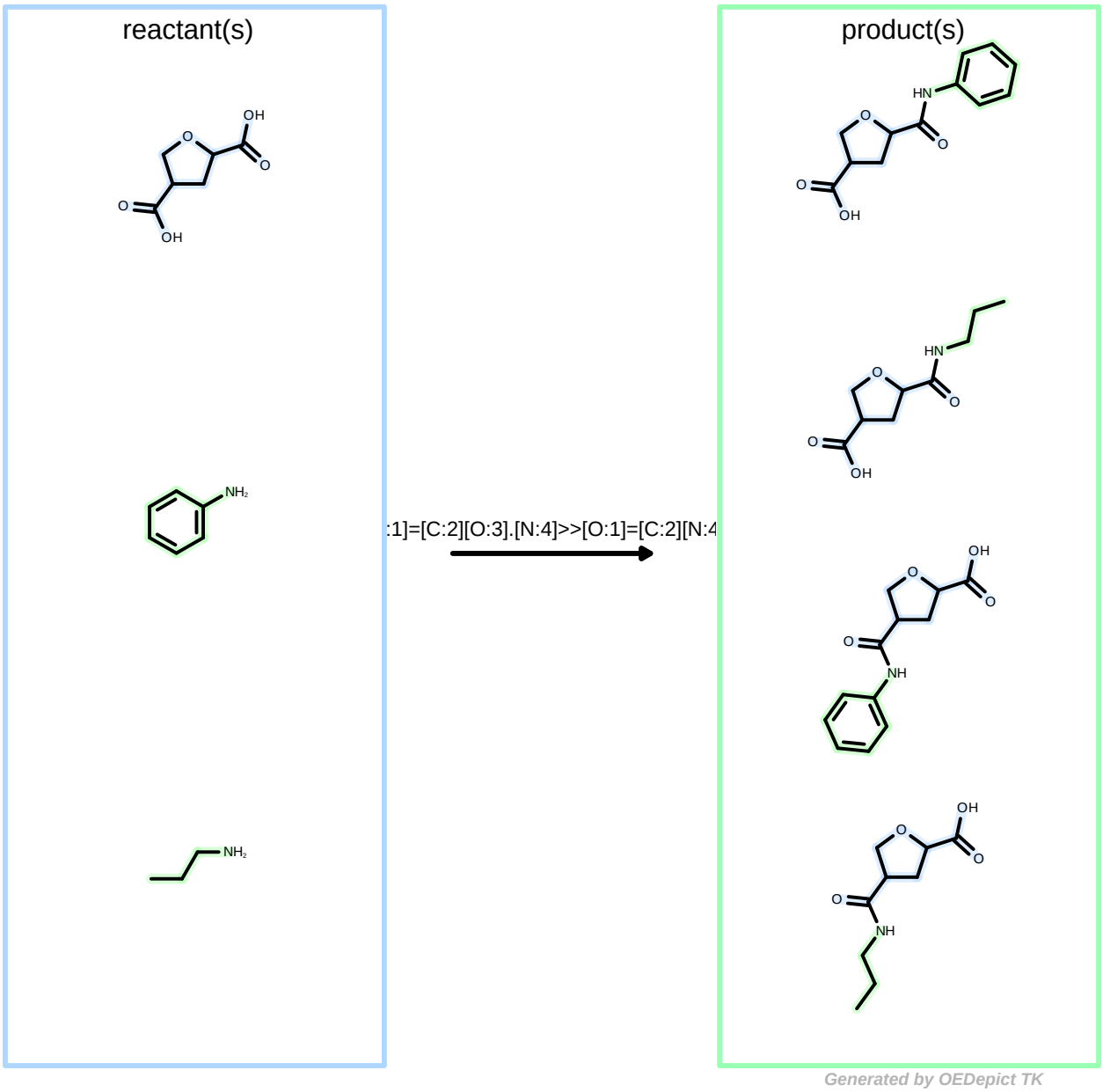


Fig. 47: **Library generation**
Example of library generation. 'OELibraryGen' enumerates all possible reacting pattern combinations to generate products.

Reactions are represented in **OEChem TK** as query molecules (`OEQMolBase`). Sets of chemical transform operations are derived from reaction molecules by differences between the reactant and product patterns and in the reaction molecule. For example, atoms and bonds that appear in the reactant pattern, but are absent in the product pattern are *destroyed*. Atoms and bonds that appear in the product pattern but not in the reactant pattern are *created*. Atoms are tracked between reactants and products by atom maps. Atom maps are stored and retrieved using the `SetMapIdx` and the `GetMapIdx` methods. Product atoms that have the same map index as reactant atoms originate from their reactant counterpart. Reactions are completely defined by a fields in a query molecule. Reaction molecules can be created from virtually any reaction file format (SMIRKS, MDL RXN, etc.), and can even be constructed programmatically.

Normalization Reactions

The *OEUniMolecularRxn* class is designed to apply a set of transformations defined by a reaction to exactly one reactant molecule. All possible transformations are applied to the initial set of atoms and bonds of an input molecule. For example, a reaction that affects a particular type of functional group will be automatically applied twice to a bi-functional molecule. The number of transformations applied by the *OEUniMolecularRxn* class is limited in order to prevent infinite loops. Consider a hypothetical reaction that methylates a methyl group. If a methyl group added in a reaction were allowed to react again, the methyl groups of a molecule would be methylated *ad infinitum*. The first protection against infinite loops provided by *OEUniMolecularRxn* is that only original atoms and bonds of the input molecule are allowed to react. Atoms and bonds created by a reaction are excluded from involvement in further reactions. A more subtle source of potential infinite loops are reactions where product atoms still match the reactant pattern after they have been involved in a chemical transformation. The *OEUniMolecularRxn* class allows a set of atoms that match a reactant pattern to react only once.

The following code demonstrates the use of the *OEUniMolecularRxn* class. *OEUniMolecularRxn* in this case is initialized using a SMIRKS pattern. The example reaction (see *Figure: Protonation*) protonates and charges an amine nitrogen. The reaction demonstrates the protection mechanisms in place to prevent under-specified reactions from causing infinite loops. The product in *Figure: Protonation* still matches the reactant pattern, however, subsequent reactions terminate when no unreacted atoms are identified.

Listing 1: Normalization

```
using System;
using OpenEye.OEChem;

public class Normalization
{
    public static int Main(string[] args)
    {
        OEUniMolecularRxn umr = new OEUniMolecularRxn("[NH2:1]>>[Nh3+:1]");

        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "NCCN");

        umr.Call(mol);

        string smi = OEChem.OEMolToSmiles(mol);
        Console.WriteLine("smiles = " + smi);
        return 0;
    }
}
```

The output of *Listing 1* is the following:

```
smiles =  C(C[NH3+])[NH3+]
```

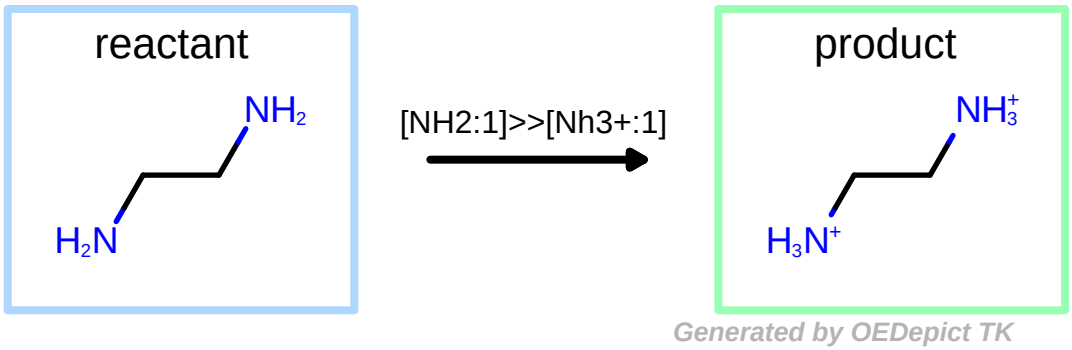


Fig. 48: **Protonation**
Example of amine nitrogen protonation.

Library Generation

The *OELibraryGen* was designed to give programmers a high degree of control when applying chemical transformations. It was also designed for efficiency. Potentially costly preprocessing is performed once before transformations can be carried out. The relative setup cost of a *OELibraryGen* instance may be high, and the memory use large as preprocessed reactants are stored in memory. Subsequent generation of products, however, is very efficient because setup costs are paid in advance. The *OELibraryGen* class serves a dual purpose of managing sets of preprocessed starting materials, and storing a list of chemical transform operations defined by a reaction molecule.

Chemical transform operations are carried out on starting materials. Starting materials provide most of the virtual matter that goes into making virtual product molecules. The *OELibraryGen* class provides an interface to associate starting materials with reactant patterns using the *SetStartingMaterial* and *AddStartingMaterial* methods. These methods associate starting materials to reactant patterns using the index (reactant number) of the pattern. In a SMIRKS pattern the first reactant (reactant number zero) is the furthest reactant on the left. Disconnected reactant patterns may be grouped into a single component using component level grouping in SMIRKS, denoted by parentheses.

Initialization with SMIRKS

OELibraryGen objects can be directly initialized with a SMIRKS pattern or an *OEQMolBase* object. See example codes in *Listing 2* – *Listing 5*.

A boolean argument is used to specify whether the SMIRKS string should be interpreted using strict SMIRKS semantics. See the SMIRKS definition describing the strict SMIRKS semantics. Passing a boolean value of false to the second method argument will relax the strict SMIRKS restrictions. The *Init* function returns false, if *OELibraryGen* can not be initialized.

OELibraryGen objects can also be initialized from an MDL reaction query file, for more details and examples see section *MDL Reaction Query File*.

Starting Materials

The *SetStartingMaterial* and *AddStartingMaterial* methods are used to initialize the starting materials corresponding to a reaction component (reactant). The declaration of these functions are:

```
uint SetStartingMaterial(OEMolBaseIterBase, uint reacnum, bool umatch);
uint AddStartingMaterial(OEMolBaseIterBase, uint reacnum, bool umatch);
// umatch (default value = true)

uint SetStartingMaterial(OEMolBase, uint reacnum, bool umatch);
```

(continues on next page)

(continued from previous page)

```
uint AddStartingMaterial(OEMolBase, uint reacnum, bool umatch);
// umatch (default value = true)
```

An iterator over molecules or a single molecule may be passed as the first argument to these methods. A copy of each starting material is stored internally along with the atom maps obtained by performing a substructure search with the corresponding reactant pattern.

The difference between using *SetStartingMaterial* and *AddStartingMaterial* methods is that by calling *SetStartingMaterial* the starting materials set in prior calls are discarded, while the *AddStartingMaterial* methods append to the list of starting materials set in prior calls.

The second argument specifies the reactant by number, starting with zero, to which the starting materials correspond. These numbers correspond with the left to right lexical ordering of reactants in the SMIRKS. In the *Listing 2* example code, there is only one reactant pattern specified in the SMIRKS string, therefore 0 is used to add the corresponding starting materials.

The final argument is used to control the pattern matching of the reactant pattern to the starting material. If the default true value is used, then only matches that contain a unique set of atoms relative to previously identified matches are used. If the value is false, every possible match including those related by symmetry will be used. *Listing 2* and the corresponding *Figure: Non-unique search* shows an example in which two unique products are generated because the reactant pattern can be mapped twice when a non-unique substructure search is performed. For more information about substructure searches see section *Substructure Search*.

Both the *SetStartingMaterial* and the *AddStartingMaterial* methods return the number of matches found in the starting materials.

Listing 2: Library generation with non-unique pattern matching

```
using System;
using OpenEye.OEChem;

public class ReactionsNonUnique
{
  public static void Main(string[] argv)
  {
      OELibraryGen lg = new OELibraryGen();
      lg.Init("[N:3][c:4]1[n:2][c:7][c:6][c:5][n:1]1>>
↪[c:6]1[c:7][n:2][c:4]2[n:3]cc[n:1]2[c:5]1");
      lg.SetExplicitHydrogens(false);
      lg.SetValenceCorrection(true);

      OEGraphMol mol = new OEGraphMol();
      OEChem.OESmilesToMol(mol, "c1cnc(nc10)N");
      uint matches = lg.AddStartingMaterial(mol, 0, false);

      Console.WriteLine("number of matches = " + matches);
      foreach (OEMolBase product in lg.GetProducts())
      {
        string smi = OEChem.OEMolToSmiles(product);
        Console.WriteLine("product smiles = " + smi);
      }
  }
}
```

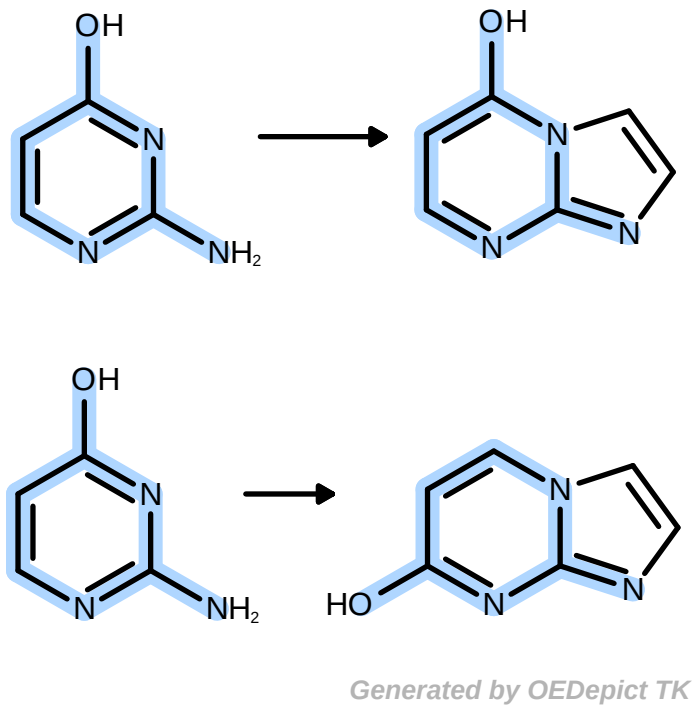


Fig. 49: **Non-unique search**
Example of generating two unique products when performing non-unique pattern matching.

The output of *Listing 2* is shown below:

```
number of matches = 2
product smiles = c1cnc2nccn2c10
product smiles = c1cn2ccnc2nc10
```

Valence Correction Modes

Once a reaction has been defined, and starting materials have been associated with each of the reactant patterns, chemical transformations can be applied to combinations of starting materials. To achieve a chemically reasonable output attention should be given to the mode of valence (or hydrogen count) correction that matches the reaction.

The *OELibraryGen* class has three possible modes of valence correction:

1. the default explicit hydrogen mode (see example in section *Explicit Hydrogen Handling*)
2. the implicit hydrogen mode (see example in section *Implicit Hydrogen Handling*)
3. the automatic valence correction mode (see example in section *Automatic Valence Correction*)

The valence correction mode must be assigned prior to adding starting materials to an *OELibraryGen* instance by calling the *SetExplicitHydrogens* and the *SetValenceCorrection* methods.

Note that the explicit hydrogen setting in effect modifies the semantics of SMIRKS. If the programmer wishes to implement strict SMIRKS according to the Daylight standard, in full, explicit hydrogens should be set on.

Explicit Hydrogen Handling

The default mode for valence correction and SMIRKS interpretation is to emulate the Daylight Reaction Toolkit. Hydrogen counts are adjusted using explicit hydrogens in SMIRKS patterns. Reactions are carried out using explicit hydrogens, and valence correction occurs when explicit hydrogens are added or deleted as defined by a reaction. The following example demonstrates strict SMIRKS and explicit hydrogen handling.

Listing 3: Library generation using explicit hydrogens

```
using System;
using OpenEye.OEChem;

public class ReactionsUsingExplH
{
    public static int Main(string[] args)
    {
        OELibraryGen libgen =
            new OELibraryGen("[O:1]=[C:2][Cl:3].[N:4][H:5]>>[O:1]=[C:2][N:4]");

        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CC(=O)Cl");
        libgen.SetStartingMaterial(mol, 0);

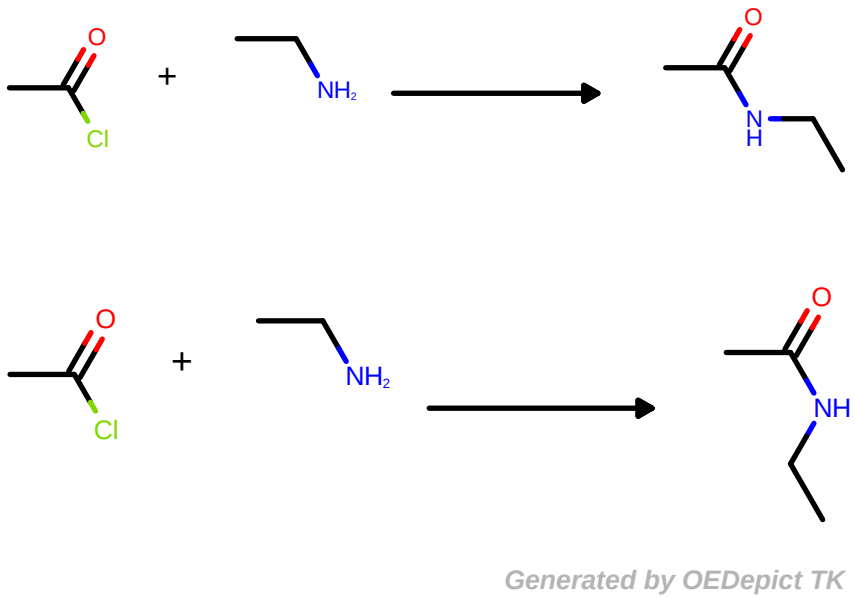
        mol.Clear();
        OEChem.OESmilesToMol(mol, "NCC");
        libgen.SetStartingMaterial(mol, 1);

        foreach (OEMolBase product in libgen.GetProducts())
        {
            string smi = OEChem.OEMolToSmiles(product);
            Console.WriteLine("product smiles = " + smi);
        }
        return 0;
    }
}
```

The output of *Listing 3* is shown below:

```
product smiles = CCNC(=O)C
product smiles = CCNC(=O)C
```

The program generates two products, each of them corresponds to the equivalent protons attached to the amine (see example in *Figure: Hydrogen handling*). If a unique set of products is desired, canonical smiles strings may be stored for verification that products generated are indeed unique.



Generated by OEDepict TK

Fig. 50: **Hydrogen handling**
Example of strict SMIRKS reaction handling with explicit hydrogen

Implicit Hydrogen Handling

The following code demonstrates how the same basic reaction given in the previous example can be carried out using the implicit hydrogen correction mode. Notice that no explicit hydrogens appear in the reaction. Instead, the SMARTS implicit hydrogen count operator appears on the right hand side of the reaction and is used to assign the implicit hydrogen count of the product nitrogen.

Listing 4: Library generation using implicit hydrogens

```
using System;
using OpenEye.OEChem;

public class ReactionsUsingImplH
{
    public static int Main(string[] args)
    {
        OELibraryGen libgen =
            new OELibraryGen("[O:1]=[C:2][Cl:3].[N:4]>>[O:1]=[C:2][Nh1:4]");
        libgen.SetExplicitHydrogens(false);

        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CC(=O)Cl");
        libgen.SetStartingMaterial(mol, 0);

        mol.Clear();
        OEChem.OESmilesToMol(mol, "NCC");
        libgen.SetStartingMaterial(mol, 1);
```

(continues on next page)

(continued from previous page)

```
        foreach (OEMolBase product in libgen.GetProducts())
        {
            string smi = OEChem.OEMolToSmiles(product);
            Console.WriteLine("product smiles = " + smi);
        }
        return 0;
    }
}
```

The output of *Listing 4* is the following:

```
product smiles = CCNC(=O)C
```

The reaction is written to work with implicit hydrogens (using the lowercase 'h' primitive), and the *OELibraryGen* instance is set to work in implicit hydrogen mode using the *SetExplicitHydrogens* method.

The *SetExplicitHydrogens* method sets the hydrogen handling mode for the *OELibraryGen* instance. *OELibraryGen* instances are constructed by default with the explicit hydrogen mode set to true. If the value is true, the *OELibraryGen* instance will add explicit hydrogens to reactant molecules when they are initialized using either the *SetStartingMaterial* or the *AddStartingMaterial* methods.

If the value is false, then the *SetStartingMaterial* and the *AddStartingMaterial* methods will suppress any explicit hydrogens in the reactant molecules, and simply retain the implicit hydrogen counts for remaining non-hydrogen atoms.

Note that the explicit hydrogen setting in effect modifies the semantics of SMIRKS. If the programmer wishes to implement strict SMIRKS according to the Daylight standard, in full, explicit hydrogens should be set on.

Automatic Valence Correction

The final example demonstrates automatic valence correction with implicit hydrogen mode. The automatic valence correction attempts to add or subtract implicit hydrogens in order to retain the valence state observed in the starting materials.

OELibraryGen instances are constructed by default with the valence correction mode being turned off. It can be turned on by passing a boolean true value to *SetValenceCorrection* method of *OELibraryGen* (see example in *Listing 5*).

When valence correction mode is enabled, the valence state for each mapped reacting atom is recorded before any chemical transformations commence. After the transform operations are complete, *OELibraryGen* will attempt to adjust (either increase or decrease) the implicit hydrogen count on atoms in the product molecule that are involved in the reaction to match the original valence state of the corresponding reactant atom. Formal charge is taken into account during the hydrogen count adjustment.

Listing 5: Library generation with automatic valence correction

```
using System;
using OpenEye.OEChem;

public class ReactionsUsingValenceCorr
{
    public static int Main(string[] args)
    {
        OELibraryGen libgen = new OELibraryGen("[O:1]=[C:2][Cl:3].[N:4]>>[O:1]=[C:2][N:4]
↪");
```

(continues on next page)

(continued from previous page)

```
        libgen.SetExplicitHydrogens(false);
        libgen.SetValenceCorrection(true);

        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CC(=O)Cl");
        libgen.SetStartingMaterial(mol, 0);

        mol.Clear();
        OEChem.OESmilesToMol(mol, "NCC");
        libgen.SetStartingMaterial(mol, 1);

        foreach (OEMolBase product in libgen.GetProducts())
        {
            string smi = OEChem.OEMolToSmiles(product);
            Console.WriteLine("product smiles = " + smi);
        }
        return 0;
    }
}
```

The output of *Listing 5* is shown below:

```
product smiles = CCNC(=O)C
```

Note that valence correction in effect modifies the semantics of SMIRKS. Thus, if the programmer wishes to implement strict SMIRKS according to the Daylight standard, in full, valence correction should be turned off.

In general, automatic valence correction is a convenience that allows straightforward reactions to be written in a simplified manner and reduces the burden of valence state bookkeeping. However, reactions that alter the preferred valence state of an atom, oxidation for example, **may not be automatically correctable**.

Reactant Tracking

The title of a product molecule is generated by concatenating reactant molecule titles with a delimiter or separator inserted between them. The default separator is the underscore character '_'. The title separator string may be changed by calling the *SetTitleSeparator* method (see example in *Listing 6*).

Listing 6: Tracing reactants

```
using System;
using OpenEye.OEChem;

public class ReactionsReactantTrace
{
    public static int Main(string[] args)
    {
        OELibraryGen libgen = new OELibraryGen("[O:1]=[C:2][Cl:3].[N:4]>>[O:1]=[C:2][N:4]
↪");
        libgen.SetExplicitHydrogens(false);
        libgen.SetValenceCorrection(true);
        uint nrproducts = 1;
        uint nrmatches  = 0;
```

(continues on next page)

(continued from previous page)

```
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CC(=O)Cl acetyl chloride");
        nrmatches += libgen.AddStartingMaterial(mol, 0);
        mol.Clear();
        OEChem.OESmilesToMol(mol, "c1ccccc1C(=O)Cl benzoyl chloride");
        nrmatches += libgen.AddStartingMaterial(mol, 0);
        nrproducts *= nrmatches;
        mol.Clear();

        nrmatches = 0;
        OEChem.OESmilesToMol(mol, "NCC ethanamine");
        nrmatches += libgen.AddStartingMaterial(mol, 1);
        mol.Clear();
        OEChem.OESmilesToMol(mol, "OCCN 2-aminoethanol");
        nrmatches += libgen.AddStartingMaterial(mol, 1);
        nrproducts *= nrmatches;
        Console.WriteLine("Number of products = " + nrproducts);

        libgen.SetTitleSeparator("#");
        foreach (OEMolBase product in libgen.GetProducts())
        {
            string smi = OEChem.OEMolToSmiles(product);
            Console.WriteLine("product smiles = " + smi + " " + product.GetTitle());
        }
        return 0;
    }
}
```

If every reactant added to an *OELibraryGen* instance has a unique identifier or name, then the title of a generated product unambiguous determines from which reactants the product was constructed. The output of *Listing 6* is shown below:

```
Number of products=4
product smiles = CCNC(=O)C acetyl chloride#ethanamine
product smiles = CC(=O)NCCO acetyl chloride#2-aminoethanol
product smiles = CCNC(=O)c1ccccc1 benzoyl chloride#ethanamine
product smiles = c1ccc(cc1)C(=O)NCCO benzoyl chloride#2-aminoethanol
```

Both *SetStartingMaterial* and *AddStartingMaterial* methods return the number of substructure matches detected in the compounds(s) added to *OELibraryGen*. Because *OELibraryGen* enumerates all possible reactant combinations to generate products, the number of products that will be generated can be calculated by multiplying the number of reactant matches. In the example in *Listing 6*, there are four expected products, because only one substructure match is detected for each reactant.

Atom Mapping

The atom mapping defines which atoms and bonds remain the same and which are altered on the reactant and products sides of the reaction.

1. Atoms that appear in both sides of the reaction are left intact.
2. Atoms and bonds that appear in the product pattern(s) but not in the reactant pattern(s) are created.
3. Atoms and bonds that disappear in the reactant pattern(s) but not in the product pattern(s) are deleted. When an

atom is deleted all of its unmapped explicit hydrogens are automatically removed.

4. Unmapped reactant atoms are deleted only in strict semantics.

The user should carefully consider whether to add a mapping or to leave unmapped an atom in the reaction. Consider, for example, the first reactant pattern in *Figure: Incorrect reaction mapping (top)* that can be mapped both to aldehydes or ketones. Because the carbon atom of the carbonyl (mapping index 2) appears in both side of the reaction, it remains intact during the reaction with all of its neighbor atoms that are not described in the reactant pattern. As a result, when the reactant pattern is mapped to an acyl chloride (see *Figure: Incorrect reaction mapping (bottom)*) the chlorine atom is still bonded to the carbon in the generated product.

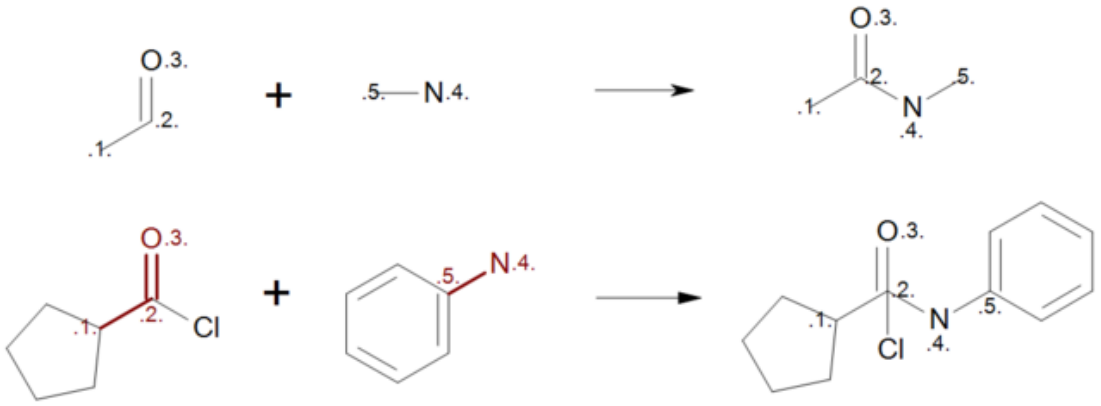


Fig. 51: **Incorrect reaction mapping**
Example of incorrect reaction mapping; (top) reaction definition, (bottom) product generation.

By removing the atom mapping from the carbon atom of the carbonyl (*Figure: Correct reaction mapping (top)*), the carbon atom will be created on the product side during the product generation. As a result, when the reactant pattern is mapped to an acyl chloride (see *Figure: Correct reaction mapping (bottom)*), the bond between the carbon and the chlorine atom is broken.

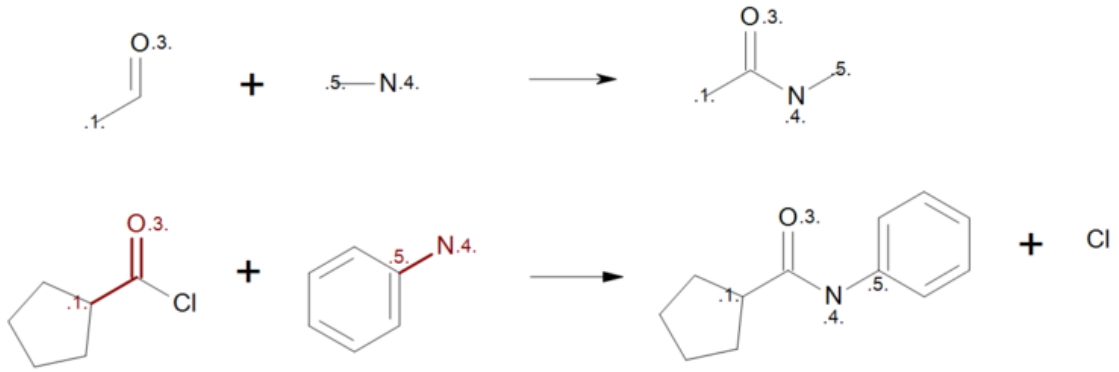


Fig. 52: **Correct reaction mapping**
Example of correct reaction mapping; (top) reaction definition, (bottom) product generation.

The `SetRemoveUnmappedFragments` method of `OELibraryGen` controls whether to keep or remove the unmapped components on the product side after the reaction is executed. When it is enabled, by calling

`SetRemoveUnmappedFragments` with the boolean value of 'true', any component on the product side that has no mapped atom is automatically removed. In case of the example in *Figure: Correct reaction mapping (bottom)*, this means deleting the isolated chlorine atom. The default behavior is to keep all components.

Figure: Incomplete reaction (top) shows an example, where not all ring atoms in the reactants are mapped. During the product generation, the unmapped atoms on the reactant side are deleted, while the unmapped atoms on the products side are created. As a result, ring substitutions are removed from the generated product in all positions where atom mapping is missing (see generated product and the removed ring substitution in *Figure: Incomplete reaction (bottom)*). The unmapped fragment (CCN) can be automatically removed from the final product by calling `SetRemoveUnmappedFragments` with the boolean value of 'true'.

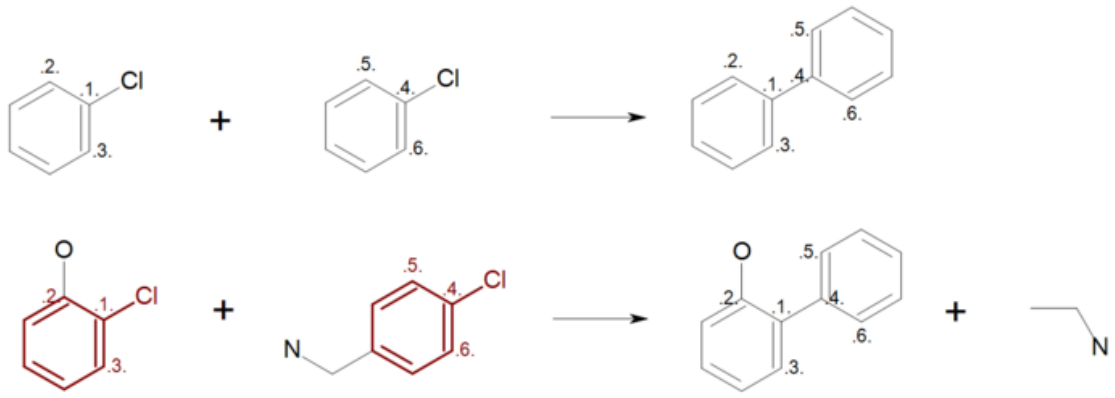


Fig. 53: **Incomplete reaction**
Example of incomplete reaction ring mapping; (top) reaction definition, (bottom) product generation.

By mapping all ring atoms in the reactants (see *Figure: Complete reaction (top)*), all ring substitutions are left intact in the generated product (see *Figure: Complete reaction (bottom)*).

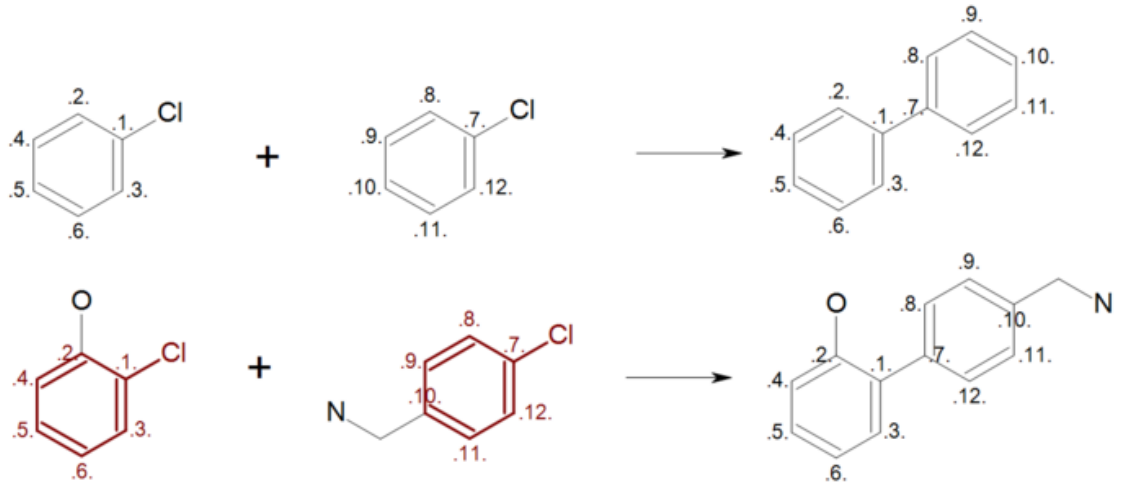


Fig. 54: **Complete reaction**
Example of complete reaction ring mapping; (top) reaction definition, (bottom) product generation.

MDL Reaction Query File

An MDL reaction query file is comprised of a series of structural data with various atom and bond query features to define each reactant and product of a reaction. Mapping indices on the atom blocks define which reactant atom becomes which product atom through the reaction.

See section *Supported MDL Query Features* for the supported MDL query features in **OEChem TK** and https://discover.3ds.com/ctfile-documentation-request-form for the MDL reaction file format.

Listing 7 shows an example how to initialize an *OELibraryGen* object from MDL reaction query. The reaction defined in the 'amide.rxn' file is shown *Figure: MDL query*.

Listing 7: Library generation using a MDL reaction query

```
using System;
using OpenEye.OEChem;

public class ReactionsRXN
{
    public static void PrintAtoms(string label, OEQAtomBaseIter aiter)
    {
        Console.WriteLine(label);

        aiter.ToFirst();
        foreach (OEAtomBase atom in aiter)
        {
            Console.Write(" atom: " + atom.GetIdx() + " ");
            Console.Write(OEChem.OEGetAtomicSymbol(atom.GetAtomicNum()));
            Console.Write(" in component " + atom.GetIntData(OEProperty.Component));
            Console.WriteLine(" with map index " + atom.GetMapIdx());
        }
    }

    public static int Main(string[] args)
    {
        oemolistream rfile = new oemolistream("amide.rxn");
        OEQMol reaction = new OEQMol();
        // reading reaction
        uint opt = OEMDLQueryOpts.ReactionQuery | OEMDLQueryOpts.SuppressExplicitH;
        OEChem.OEReadMDLReactionQueryFile(rfile, reaction, opt);
        PrintAtoms("Reactant atoms:", reaction.GetQAtoms(new OEAtomIsInReactant()));
        PrintAtoms("Product atoms :", reaction.GetQAtoms(new OEAtomIsInProduct()));
        // initializing library generator
        OELibraryGen libgen = new OELibraryGen();
        libgen.Init(reaction);
        libgen.SetExplicitHydrogens(false);
        libgen.SetValenceCorrection(true);
        // adding reactants
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccc(cc1)C(=O)O");
        libgen.AddStartingMaterial(mol, 0);
        mol.Clear();
        OEChem.OESmilesToMol(mol, "CC(C)CN");
        libgen.AddStartingMaterial(mol, 1);
```

(continues on next page)

(continued from previous page)

```
        // accessing generated products
        foreach (OEMolBase product in libgen.GetProducts())
        {
            Console.WriteLine("product smiles = " + OEChem.OEMolToSmiles(product));
        }
        return 0;
    }
}
```

The output of *Listing 7* is the following:

```
Reactant atoms:
 atom: 0 C in component 1 with map index 1
 atom: 1 O in component 1 with map index 0
 atom: 2 O in component 1 with map index 2
 atom: 3 C in component 2 with map index 4
 atom: 4 N in component 2 with map index 3
Product atoms :
 atom: 5 C in component 3 with map index 1
 atom: 6 N in component 3 with map index 3
 atom: 7 C in component 3 with map index 4
 atom: 8 O in component 3 with map index 2
product smiles = CC(C)CNC(=O)c1ccccc1
```

OEReadMDLReactionQueryFile function reads the MDL reaction query directly into a *OEQMolBase* object, which then can be used to initialize an *OELibraryGen* object.

```
OEChem.OEReadMDLReactionQueryFile(rfile, reaction, opt);
```

The user can iterate over the reactant and product atoms of a reaction using the *OEAtomIsInReactant* and the *OEAtomIsInProduct* functors, respectively.

```
PrintAtoms("Reactant atoms:", reaction.GetQAtoms(new OEAtomIsInReactant()));
PrintAtoms("Product atoms :", reaction.GetQAtoms(new OEAtomIsInProduct()));
```

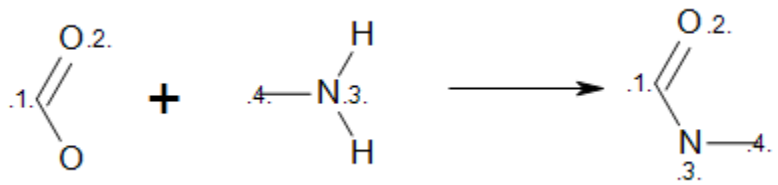


Fig. 55: **MDL Query**
Example of MDL query reaction

The reaction 'role' property of an atom can also be accessed with the *GetRxnRole* method of the *OEAtomBase* class.

The *GetMapIdx* method returns the atom map index read from the reaction file.

```
Console.WriteLine(" with map index " + atom.GetMapIdx());
```

An unmapped atom has a map index zero. For more information about these functions and the *OERxnRole* namespace, please refer to the *OEChem API* manual.

Since the explicit hydrogens are suppressed when the reaction is converted into a query molecule,

```
uint opt = OEMDLQueryOpts.ReactionQuery | OEMDLQueryOpts.SuppressExplicitH;
```

the explicit hydrogens in the starting materials can also be suppressed, yielding faster performance while still generating the same products.

```
libgen.SetExplicitHydrogens(false);
```

More details on suppressing hydrogens in a query can be found in section *Explicit Hydrogens*. It is also recommended to set the valence correction mode on.

```
libgen.SetValenceCorrection(true);
```

The MDL reaction query can also be read into a *OEMolBase* object (see code snippet below). In this case, the *OEReadMDLReactionQueryFile* function attaches the query features present in the input MDL file to the related atoms and bonds of the *OEMolBase* object. The *OEQMolBase* object can be subsequently created by calling the *OEBuildMDLQueryExpressions* function with *OEMDLQueryOpts.ReactionQuery* option.

```
OEGraphMol reaction = new OEGraphMol();
OEChem.OEReadMDLReactionQueryFile(rfile,reaction);
// reaction can be manipulated here
OEQMol qreaction = new OEQMol();
// build OEQMol with OEMDLQueryOpts.ReactionQuery option
OEChem.OEBuildMDLQueryExpressions(qreaction,reaction,OEMDLQueryOpts.ReactionQuery);
OELibraryGen libgen = new OELibraryGen();
libgen.Init(qreaction);
```

The declarations of these functions are:

```
bool OEReadMDLReactionQueryFile(oemolistream ifs, OEMolBase mol);
bool OEReadMDLQueryFile(oemolistream ifs, OEQMolBase qmol, uint opts);
// opts (default value = OEMDLQueryOpts.ReactionQuery)
bool OEBuildMDLQueryExpressions(OEQMolBase qmol, OEMolBase mol, uint opts);
// opts (default value = OEMDLQueryOpts.Default)
```

The 'opts' parameter defines how the MDL query is interpreted when *OEQMolBase* object is constructed.

For more details on the various options in the *OEMDLQueryOpts* namespace and the MDL query based substructure search, please see chapter *Substructure Search with MDL Queries*.

Hydrogens on the Reactant Side

When defining a reaction it is very important to consider not only correct atom mapping but also to anticipate the different kind of starting materials the reactant pattern(s) will match.

At first glance, the reaction shown in *Figure: Missing hydrogen (reaction)* seems correct, however some of the generated products have carbon atom with incorrect 5 valences see example 'A' in *Figure: Missing hydrogen (products)*.

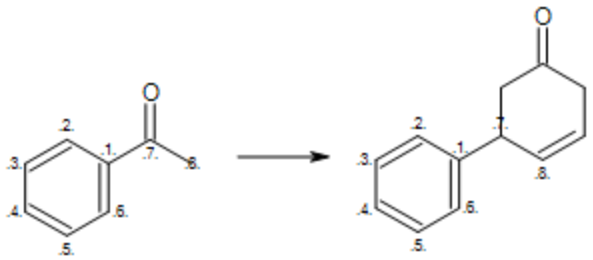


Fig. 56: **Missing hydrogen (reaction)**
Example of missing explicit/implicit hydrogen in reactant (equivalent SMIRKS: [c:2]1[c:3][c:4][c:5][c:6][c:1]1([C:7](=O)[C:8])>>[c:2]1[c:3][c:4][c:5][c:6][c:1]1([C:7]2[C:8]=CCC(=O)C2)

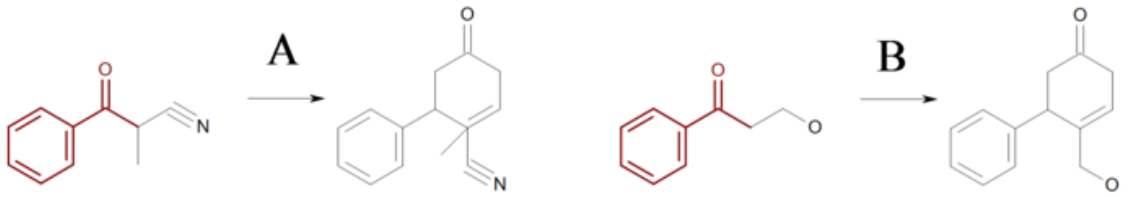


Fig. 57: **Missing hydrogen (products)**
Example of products generated with reaction defined in 'Missing hydrogen (reaction)'

By adding two explicit (or implicit) hydrogens to the reactant atom with atom map 8 in *Figure: Missing hydrogen (reaction)*, ensures that only reaction 'B' is executed, since the starting material of reaction 'A' no longer matches the reaction pattern.

Note

In order to generate products with correct valences, a mapped reactant atom should have as many as explicit or implicit hydrogens as many as new bonds the atom forms on the product side

Product Kekulization

Prior to returning a generated product, `OELibraryGen` calls the `OEKekulize` and `OEAssignAromaticFlags` functions to determine the Kekulé form and aromaticity of the generated product. Since `OEKekulize` requires the correct specification of the formal charges and implicit hydrogen counts of each atom, the Kekulé form of a generated product depends on the reaction specification.

The three reactions shown in *Identical transformations* describe the same set of transformations, but they generate products with different but valid Kekulé forms. Reactions 'A' and 'B' in *Identical transformations* will produce products 'A' and 'B' in *Products*, respectively. However, reaction 'C' arbitrarily generates either product 'A' or its tautomer product 'B', since this reaction describes ambiguous Kekulé form for the 5-membered ring in the product.

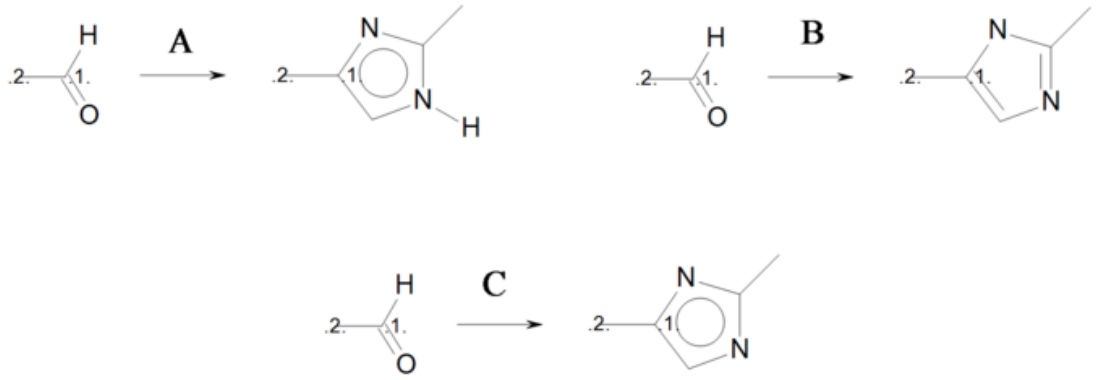


Fig. 58: **Identical transformations**
Three reactions describing the same transformation

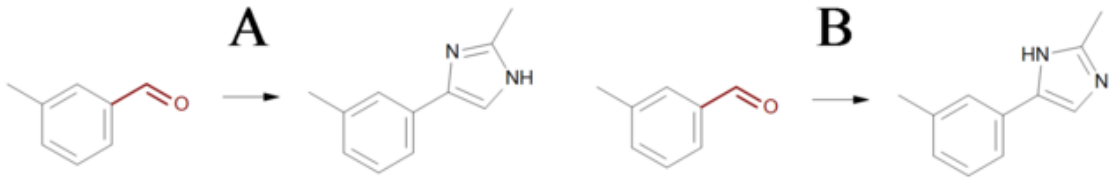


Fig. 59: **Products**
Example of products generated with reactions defined in 'Identical transformations'

Note

In order to generate products with the desired Kekulé form, either the product has to have only specific bonds or explicit/implicit hydrogens have to be specified to eliminate ambiguities.

Even though the reaction in *Reaction* defines an unambiguous Kekulé form, this form is not suited for every product generated by the reaction (see examples in *Products*). In order to ensure that the `OELibraryGen` generates products with valid Kekulé form the following steps are performed. First, Kekulization is performed based on the reaction specification. If it is unsuccessful, *i.e.* `OEKekulize` function returns false, alternatives are tried by adding and removing implicit hydrogens from aromatic nitrogens, until a valid Kekulé form is found.



Fig. 60: **Reaction**
Example of reaction with generic atoms in an aromatic ring system (``s*`` = number of non-hydrogen substituents as drawn).

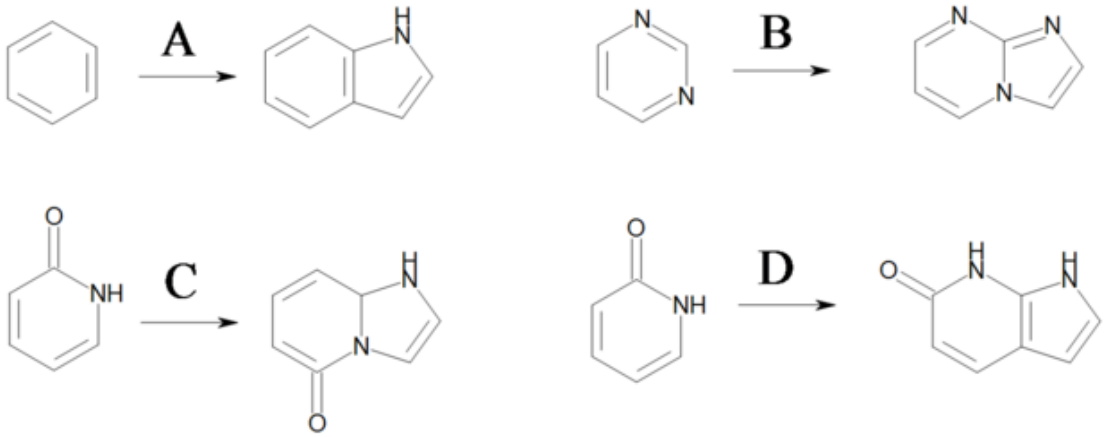


Fig. 61: **Products**
Example of products generated with the reaction shown in previous figure

Advanced OEChem

2.1.20 Multi-conformer Molecules

OEMols

Up to this point in the manual, all of the examples have involved using concrete *OEGraphMol* molecules. These molecules have been utilizing the functionality defined in the API of the *OEMolBase* abstract base-class. At this point we will introduce another layer of abstraction in **OEChem TK**'s representation of molecules. In **OEChem TK**, we draw a distinction between molecules that are limited to a single conformer and those that may have any number of conformers. While this may be an arbitrary decision, it is a pragmatic one which allows more efficient implementation of both classes. The single-conformer molecule's API is defined by the already familiar *OEMolBase* abstract base-class. The multi-conformer molecule's API is defined by another abstract base-class, the *OEMCMolBase* (here the *MC* stands for Multi-Conformer). The *OEMCMolBase* class inherits publicly from *OEMolBase*, thus the multi-conformer molecule supports the single-conformer API but adds additional functions to manage conformers. Both the single-conformer and the multi-conformer molecules contain atoms and bonds, but only the multi-conformer molecule contains conformers as first-class objects.

> **Note**
>
> *OEMCMolBase* conformer coordinates can be stored in *OEHalfFloat* (16-bit), `float` (32-bit), `double` (64-bit), or `long double` (>= 64-bit). The precision is determined by the constructor argument given to *OEMol*, taken from the *OEMCMolType* namespace. However, `float` is the default implementation as a space optimization and for historical purposes.

The *OEGraphMol* is a concrete class that supports the *OEMolBase* API and can be passed to functions which take an *OEMolBase* as an argument. The *OEMol* is a concrete class that supports the *OEMCMolBase* API in addition to the *OEMolBase* API. Therefore, an *OEMol* can be passed to any function which takes either an *OEMolBase* or an *OEMCMolBase* as an argument.

Since an *OEMCMolBase* is-a *OEMolBase* all the same powerful graph functions can be applied to the *OEMCMolBase*. This powerful type of abstraction allows for pure graph algorithms to still be applied to multi-conformer molecules without requiring the algorithms to know that they are operating on a different molecule representation. When the algorithm should be changed to accommodate multiple conformers the function can be overloaded to provide the different functionality for *OEMCMolBase*.

For example, the behavior in the following are slightly different based upon whether the molecule being passed is an *OEMolBase* or an *OEMCMolBase*.

- *OEAddExplicitHydrogens*
- *OECenter*
- *OEReadMolecule*
- *OERMSD*
- *OERotate*
- *OESubsetMol*
- *OETranslate*
- *OEWriteMolecule*

There are also functions that only make sense for multi-conformer molecules. In this case an overload for *OEMolBase* is simply not provided. This prevents users from inadvertently passing a single-conformer molecule when a multi-conformer molecule is required. An example of this would be in the OEOmega toolkit which is designed to generate multi-conformer molecules, hence it requires an *OEMCMolBase* for the conformers to be output to.

Conformers

An *OEMCMolBase* contains one or more conformers. These conformers are managed in a manner very similar to *atoms* and *bonds*. Conformers can only be created or destroyed in the context of an *OEMCMolBase*, and must be accessed via member functions laid out in that API. Conformers are represented by the abstract base-class *OEConfBase*. The *GetActive* and *SetActive* methods are often sufficient for accessing conformations in multi-conformer molecules, but alternate access methods are also provided, e.g., *GetConf* and *GetConfs*.

It is sometimes convenient to be able to treat a conformer object as its own single-conformer molecule. For this reason, *OEConfBase* inherits from *OEMolBase*. Therefore, although a conformer is contained within a multi-conformer molecule, it can act as a single-conformer molecule, and can be passed to functions that have an *OEMolBase* argument.

One must be cautious when utilizing this *OEMolBase* inheritance functionality. Each multi-conformer molecule has only a single heavy-atom graph. For functions which query the graph portion of a molecule, a conformer will reflect the graph properties of its parent multi-conformer molecule. Graph properties include the connection table of atoms and bonds, as well as any properties stored by the atoms and bonds. A conformer is only independent of its parent for non-graph (*e.g.* conformational) properties. The logical extension of this principle is that changes made to the graph properties of one conformer will effect its parent multi-conformer molecule and thus all the other conformers in that

molecule as well. The sharing of a common connection table prevents tautomers from being modeled together with an *OEMCMolBase*.

See also

See *Design Decisions* for a complete inheritance graph explaining the relationship of **OEChem TK** molecules.

Coordinate information is stored by the conformer, not the shared atoms and bonds, which allows the conformers to share the same heavy-atom graph but have different spatial configurations. In **OEChem TK**, these conformers are represented by *OEConfBases*, which are first-class objects. The conformer is the only additional property presented by the *OEMCMolBase*. Access to conformers is similar to *atoms and bonds*, as shown by the following table.

Description	Return Type	Method	See Also
Number of Conformers	unsigned int	*NumConfs*	
Maximum Conformer Index	unsigned int	*GetMaxConfIdx*	*Atom, Bond, and Conformer Indices*
Access to a Conformer	*OEConfBase*	*GetConf*	*Predicate Functors*
Access to all Conformers	*OEIterBase*	*GetConfs*	*Atom and Bond Traversal*
Create a new Conformer	*OEConfBase*	*NewConf*	*Atom and Bond Creation*
Remove a Conformer	bool	*DeleteConf*	
Rearrange Conformers	bool	*OrderConfs*	

Conformer Iteration

Conformers of an *OEMCMolBase* can be iterated over the same way *atoms and bonds* of an *OEMolBase* are iterated over. *GetConfs* returns an iterator over the conformers of the molecule. This allows one to have multiple conformation objects at once and to treat the *OEMCMolBase* as a container of single-conformer molecules.

Listing 1 demonstrates the use of the conformers as first class objects. Each conformer is represented by an *OEConfBase* which inherits from *OEMolBase*. Thus, each conformer can be treated as an independent molecule with respect to its coordinates as shown in the example code below.

Listing 1: Retrieving the bounding box for a set of conformers

```
using System;
using OpenEye.OEChem;

public class ConfBoundingBox
{
    public static int Main(string[] args)
    {
        oemolistream ifs = new oemolistream(args[0]);
        float[] ctr = new float[3];
        float[] ext = new float[3];
        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            float[] box = {
                float.PositiveInfinity,
                float.PositiveInfinity,
                float.PositiveInfinity,
```

(continues on next page)

(continued from previous page)

```
                float.NegativeInfinity,
                float.NegativeInfinity,
                float.NegativeInfinity
            };

            foreach (OEConfBase conf in mol.GetConfs())
            {
                OEChem.OEGetCenterAndExtents(conf, ctr, ext);
                box[0] = Math.Min(box[0], ctr[0] - ext[0]);
                box[1] = Math.Min(box[1], ctr[1] - ext[1]);
                box[2] = Math.Min(box[2], ctr[2] - ext[2]);
                box[3] = Math.Max(box[3], ctr[0] + ext[0]);
                box[4] = Math.Max(box[4], ctr[1] + ext[1]);
                box[5] = Math.Max(box[5], ctr[2] + ext[2]);
            }

            Console.WriteLine("Bounding box for the conformers of " + mol.GetTitle());
            Console.Write("Lower Extent: {0:0.000} {1:0.000} {2:0.000}", box[0], box[1],␣
↪box[2]);
            Console.Write(" Upper Extent: {0:0.000} {1:0.000} {2:0.000}", box[3], box[4],
↪ box[5]);
            Console.WriteLine();
        }
        return 0;
    }
}
```

OEGetCenterAndExtents takes an *OEMolBase* as an argument, not an *OEConfBase*. However, it is still usable on each conformer since *OEConfBase* inherits from *OEMolBase*. Therefore, all the functions that were written for *OEMolBase* are automatically usable on conformers.

> **Note**
>
> **OEChem TK** makes the same guarantees on conformers of a multi-conformer molecule as it does for atoms and bonds of a molecule. That is, conformer objects and their indices are stable across any other method call, with the exception of *SweepConfs*.

Conformer Creation

The most common method to create conformers in a molecule is by reading a molecule from a file (see section *Input and Output*). However, when manipulating molecules it is often necessary to create conformers on-the-fly. In **OEChem TK**, this is done with the *NewConf* method. There are numerous overloads of *NewConf*. All of the overloads create conformers with the capacity to store coordinates for the current number of atoms in the molecule. *NewAtom* adjusts this capacity as necessary. The default *OEMCMolBase* constructor puts the molecule in a state with a single empty conformer (as does the *Clear* method).

See also

Parallel Data Structures for a discussion of how indices are used to index into coordinate arrays.

Listing 2 demonstrates how to generate a multi-conformer water molecule from scratch. It then measures the hydrogen-oxygen-hydrogen angle of the two conformers. Remember, the **OEChem TK** definition of conformer is very loose, any set of Cartesian coordinates constitutes another conformer. In this case the configuration of the atoms to each other is the same, the location of the molecule has just changed. No effort is given to prevent the user from creating duplicate conformers.

Listing 2: Creating conformers from explicit sets of coordinates

```
using System;
using OpenEye.OEChem;

public class NewConfFromCoords
{
    public static double ToDegrees(double radians)
    {
        double degrees = (180 / Math.PI) * radians;
        return degrees;
    }

    public static int Main(string[] args)
    {
        OEMol mol = new OEMol();

        OEAtomBase o  = mol.NewAtom(OEElemNo.O);
        OEAtomBase h1 = mol.NewAtom(OEElemNo.H);
        OEAtomBase h2 = mol.NewAtom(OEElemNo.H);

        float[] acrds = {
                         0.0f, 0.0f, 0.0f,
                         0.9584f, 0.0f, 0.0f,
                        -0.2392f, 0.9281f, 0.0f
                        };

        // Grab the default conformer
        OEConfBase acnf = mol.GetConf(new OEHasConfIdx(0));

        acnf.SetCoords(acrds);

        float[] bcrds = {
                         0.0f, 0.0f, 0.0f,
                         0.9584f, 0.0f, 0.0f,
                        -0.2392f, -0.9281f, 0.0f
                        };

        OEConfBase bcnf = mol.NewConf(bcrds);

        double aang = ToDegrees(OEChem.OEGetAngle(acnf, h1, o, h2));
        Console.WriteLine("1st Water Angle: " + aang);
```

(continues on next page)

(continued from previous page)

```
        double bang = ToDegrees(OEChem.OEGetAngle(bcnf, h1, o, h2));
        Console.WriteLine("2nd Water Angle: " + bang);
        return 0;
    }
}
```

Warning

An *OEMCMolBase* is constructed with one conformer already present. Special care must be taken in *Listing 2* to insert the coordinates of the first conformer into this conformer.

There is also a version of *NewConf* which takes an *OEMolBase* and copies the coordinates of the passed molecule into the new conformer. *NewConf* is expecting that the molecule passed has the same graph as the *OEMCMolBase* which is the parent of the new conformer. It is important to note that this version of *NewConf* can take any instance of an *OEMolBase*, such as an *OEGraphMol* or an *OEMol*. When an *OEMol* is passed to *NewConf*, the coordinates of the newly created conformer will come from the first conformation of the molecule passed.

Listing 3 demonstrates how to use *NewConf* with another conformer as the argument. The purpose of this code is to only copy conformers within an arbitrary energy cutoff from the `src` to the `dst` molecule.

Listing 3: Filtering conformers based upon energy cutoff

```
using System;
using OpenEye.OEChem;

public class FilterConfs
{
    public static void Main(string[] argv)
    {
        oemolistream ifs = new oemolistream(argv[0]);
        oemolostream ofs = new oemolostream(argv[1]);

        OEMol src = new OEMol();
        while (OEChem.OEReadMolecule(ifs, src))
        {
            OEMol dst = new OEMol(src.SCMol());
            dst.DeleteConfs();

            foreach (OEConfBase conf in src.GetConfs())
            {
                if (conf.GetEnergy() < 25.0f)
                {
                    dst.NewConf(conf);
                }
            }

            if (dst.NumConfs() > 0)
            {
                OEChem.OEWriteMolecule(ofs, dst);
            }
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

The `dst` molecule is copy constructed from the single-conformer representation (the first conformer) of the `src` molecule. This is to grab the molecule graph information and any other non-conformational information. Then *`DeleteConfs`* is called to get rid of the first conformer. It may be copied back in the conformer loop later in the code if it meets the energy criteria.

> **Warning**
>
> An *`OEMCMolBase`* with no conformers can be dangerous to pass to other **OEChem TK** functions, so the number of conformers is checked before calling *`OEWriteMolecule`*.

Input and Output

Molecule streams can read both single and multi-conformer molecules from any file format. Many of the file formats supported by **OEChem TK** are inherently a single conformer format (SDF and MOL2, for example). However, a common practice is to store multiple conformers in these files. **OEChem TK** supports a rather advanced mechanism for recovering these separate conformers into a single, multi-conformer *`OEMCMolBase`*. Note that this does not apply to file formats where conformers are stored together, OEBinary (.oeb), for example. OEBinary files store either single-conformer or multi-conformer molecules explicitly, so the file itself determines how to deal with conformers. Additionally, file formats that have no notion of conformers (*i.e.* SMILES) are unaffected by this feature.

The *`oemolistream.SetConfTest`* method sets a functor that is used to compare the graphs of incoming molecules in order to determine whether to combine them. These functors are subclasses of *OEConfTestBase*. Several predefined versions include:

OEDefaultConfTest
: The *OEDefaultConfTest* never combines connection tables into multi-conformer molecules.

OEIsomericConfTest
: The *OEIsomericConfTest* combines subsequent connection tables into a multi-conformer molecule if they:

  1. Have the same title (optional)
  2. Have the same numbers of atoms and bonds in the same order
  3. Each atom and bond must have identical properties with its order correspondent in the subsequent connection table
  4. Have the same atom and bond stereochemistry

  No changes are made to the connection table.

  The constructor for *OEIsomericConfTest* has a default argument for whether or not to compare titles. If the constructor is called with no arguments or with the argument `true`, the titles will be required to be the same. Otherwise, the titles will not be compared. In the latter instance, each conformer will have the individual title of its original connection table and the multi-conformer molecule will reflect the title of the active conformer.

OEOmegaConfTest
: The *OEOmegaConfTest* is almost exactly the same as *OEIsomericConfTest* with the exception that differences in invertible nitrogens are disregarded. Therefore, invertible nitrogens with different stereo chemistry are considered conformers of the same molecule.

This definition is meant to be exactly in line with what **Omega** will generate. Therefore, *OEOmegaConfTest* can be used on non-OEB files to generated with **Omega** to recover the exact set of conformers **Omega** generated for a particular molecule.

OEAbsoluteConfTest

The *OEAbsoluteConfTest* combines subsequent connection tables into a multi-conformer molecule if they:

1. Have the same title (optional)
2. Have the same number of atoms and bonds in the same order
3. Each atom and bond must have identical properties with its order correspondent in the subsequent connection table

This conformer test sets all fully specified isomeric values to UNDEFINED.

The constructor for *OEAbsoluteConfTest* has a default argument for whether or not to compare titles. If the constructor is called with no arguments or with the argument true, the titles will be required to be the same. Otherwise, the titles will not be compared. In the latter instance, each conformer will have the individual title of its original connection table and the multi-conformer molecule will reflect the title of the active conformer.

OEAbsCanonicalConfTest

The *OEAbsCanonicalConfTest* combines subsequent connection tables into a multi-conformer molecule if they:

1. Have the same absolute (non-isomeric) graph

This conformer test puts all of the molecules in their canonical atom order. In addition, all fully specified isomeric values are set to UNDEFINED.

Listing 4: Reading in multi-conformer molecules from single-conformer files

```
using System;
using OpenEye.OEChem;

public class ReadInMultiConf
{
    public static int Main(string[] args)
    {
        oemolistream ifs = new oemolistream(args[0]);
        ifs.SetConfTest(new OEIsomericConfTest());

        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            Console.WriteLine(mol.GetTitle() + " has " +
                              mol.NumConfs() + " conformers");
        }
        return 0;
    }
}
```

Listing 4 will read multi-conformer molecules from an input file based on *OEIsomericConfTest*. This assumes that the conformers are ordered next to each other in the input file.

Note

The *OEIsomericConfTest* constructor can be passed `false` to allow conformers to be combined when they have different titles. This is very useful when dealing with files created by programs that modify molecule titles to indicate conformer number (*i.e.* acetsali_1, acetsali_2, acetsali_3). The modified titles are accessible through the `GetTitle` method on each individual `OEConfBase`.

Dude, where's my SD data?

SD tag data can be added to anything that derives from *OEMolBase*, including `OEMCMolBase` or `OEConfBase`. Generally, **OEChem TK** will never lose any data when reading or writing. However, there are constraints placed on **OEChem TK** as to where the SD data must go based upon the file format being used.

An ambiguity occurs when adding SD tag data to an `OEMCMolBase` and then writing it to SDF. SDF files do not support multiple conformers. However, **OEChem TK** can automatically read consecutive conformers out of a SDF file into a `OEMCMolBase`. To preserve the SD data **OEChem TK** has no choice but to push the data onto the conformers.

OEB files do not have this restriction upon them because they do support multi-conformer molecules. The following table shows how to round-trip SD tag data through the SDF and OEB formats.

Attached To	Written To	Read Into	Attached To
`OEMCMolBase`	*SDF*	`OEMCMolBase`	`OEConfBase`
`OEMCMolBase`	*SDF*	*OEMolBase*	
`OEMCMolBase`	*OEB*	`OEMCMolBase`	
`OEMCMolBase`	*OEB*	*OEMolBase*	
`OEConfBase`	*SDF*	`OEMCMolBase`	`OEConfBase`
`OEConfBase`	*SDF*	*OEMolBase*	
`OEConfBase`	*OEB*	`OEMCMolBase`	`OEConfBase`
`OEConfBase`	*OEB*	*OEMolBase*	
OEMolBase	*SDF*	`OEMCMolBase`	`OEConfBase`
OEMolBase	*SDF*	*OEMolBase*	
OEMolBase	*OEB*	`OEMCMolBase`	`OEConfBase`
OEMolBase	*OEB*	*OEMolBase*	

Note

The `OEFormat.CSV` format behaves identical to the `OEFormat.SDF` format for how SD data is handled.

Practically, it is best to never attach SD tag data to an `OEMCMolBase`. This should only be done as a space optimization when it is assured that the multi-conformer molecule will only be written to OEB.

To this end, when an *OEMol* copy constructs from an *OEGraphMol* the SD tag data is attached to the first conformer.

See also

- *Generic Data* chapter
- *SD Tagged Data Manipulation* section
- *PDB Tagged Data Manipulation* section

2.1.21 Molecular Database Handling

The *OEMolDatabase* class provides a fundamentally different abstraction over a molecule file than the combination of *oemolistream* and *OEReadMolecule*. The central underlying principle utilized by this class is that many operations can be performed on a molecular file without requiring the overhead of fully parsing the molecule record into an *OEMolBase* object. Instead, we can think of a molecular file as a database that can be manipulated with much cheaper operations than *OEReadMolecule* and *OEWriteMolecule*.

Opening and Reading

OEMolDatabase objects provide the ability to access any molecule in a molecular database file in constant time, O(1). This is accomplished by paying the overhead of scanning the file during the *OEMolDatabase.Open* call. However, *OEMolDatabase.Open* is designed to operate extremely fast on any molecule file format **OEChem TK** supports. *OEMolDatabase.Open* is usually limited by disk bandwidth instead of parsing and perception like *OEReadMolecule* and *OEWriteMolecule*.

After a database file is opened, the memory overhead of *OEMolDatabase* is minimal since no molecules are stored in memory. Instead, the *OEMolDatabase* only stores a 8 byte file offset for each molecule record in the file. *Listing 1* demonstrates how to utilize this feature to retrieve the "Nth" molecule from a molecule file using the *OEMolDatabase.GetMolecule* method.

Listing 1: Retrieving the Nth molecule in a file

```
using System;

using OpenEye.OEChem;

public class NthMolecule
{
    public static void Main(String[] args)
    {
        if (args.Length != 3)
            OEChem.OEThrow.Usage("NthMolecule <input> <output> <index>");

        OEMolDatabase moldb = new OEMolDatabase();
        if (!moldb.Open(args[0]))
            OEChem.OEThrow.Fatal("Unable to open " + args[0]);

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
            OEChem.OEThrow.Fatal("Unable to open " + args[1]);

        uint idx = Convert.ToUInt32(args[2]);

        OEMol mol = new OEMol();
        if (!moldb.GetMolecule(mol, idx))
            OEChem.OEThrow.Fatal("Unable to read a molecule from index " + idx);

        OEChem.OEWriteMolecule(ofs, mol);

    }
}
```

Listing 1 checks the return value of *OEMolDatabase.GetMolecule* for false, indicating the molecule record at that position in the file does not contain a valid molecule. For example, molecules without any atoms are valid records in

`.sdf` files.

> **Note**
>
> *OEMolDatabase.Open* is still a O(N) operation in order to know the position of each molecule record in the file. However, this method is significantly cheaper than using *OEReadMolecule*, instead being limited by hard disk bandwidth instead of processing speed. The *OEMolDatabase.Open* method can also be sped up by creating an associated `.idx` file as described by the *Index Files* section.

Direct Data Access

OEMolDatabase achieves much of its speed by treating molecules as chunks of bytes instead of *OEMolBase* objects. This abstraction is leaked a little bit by providing users the ability to access the raw bytes of the molecule record as well through the *OEMolDatabase.GetMolecule* method that takes a *oemolostream*. For example, the user could pass this method a *oemolostream* that has been opened with *oemolostream.openstring* in order to dump the desired bytes to an in memory buffer. *Listing 2* demonstrates how to use this feature to retrieve a subset of molecules from a database file similar to how the `LIMIT` and `OFFSET` keywords work for an SQL query.

Listing 2: Retrieving a subset of a file

```
using System;

using OpenEye.OEChem;

public class DatabaseSubset
{
    public static void Main(String[] args)
    {
        if (args.Length != 4)
            OEChem.OEThrow.Usage("DatabaseSubset <input> <output> <offset> <limit>");

        OEMolDatabase moldb = new OEMolDatabase();
        if (!moldb.Open(args[0]))
            OEChem.OEThrow.Fatal("Unable to open " + args[0]);

        String ext = OEChem.OEGetFileExtension(args[1]);
        if (moldb.GetFormat() != OEChem.OEGetFileType(ext))
            OEChem.OEThrow.Fatal("Output format does not match input format: " +
                                 OEChem.OEGetFileExtension(args[0]) + " != " + ext);

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
            OEChem.OEThrow.Fatal("Unable to open " + args[1]);

        uint offset = Convert.ToUInt32(args[2]);
        uint limit = Convert.ToUInt32(args[3]);

        uint maxIdx = offset + limit;

        for (uint idx = offset; idx < maxIdx; idx++)
        {
            moldb.WriteMolecule(ofs, idx);
```

(continues on next page)

(continued from previous page)

```
        }

    }
}
```

Note

The *oemolostream* must be set up to write output to the exact same file format that the *OEMolDatabase* is opened on. If file format conversion is also desired during the read operation, the user should user *OEMolDatabase.GetMolecule* to read the molecule into an *OEMolBase* and then use *OEWriteMolecule*.

Title Access

Molecule meta-data is often useful for manipulating databases regardless of the molecule connection table. For this reason the *OEMolDatabase* provides access to the molecule title through the *OEMolDatabase.GetTitle* method. The *OEMolDatabase.GetTitle* returns the same string that would be returned by *OEMolBase.GetTitle* if the molecule was read in with *OEReadMolecule*. The difference is that the *OEMolDatabase.GetTitle* method is more efficient because it will only parse the title from the molecule record, and skip the rest of the bytes in the molecule record. *Listing 3* demonstrates how to use *OEMolDatabase.GetTitle* to implement a more efficient version of the *molextract* example.

Listing 3: Extract molecules by title

```
using System;

using OpenEye.OEChem;

public class DatabaseExtract
{
    public static void Main(String[] args)
    {
        if (args.Length != 3)
            OEChem.OEThrow.Usage("DatabaseExtract <input> <output> <title>");

        OEMolDatabase moldb = new OEMolDatabase();
        if (!moldb.Open(args[0]))
            OEChem.OEThrow.Fatal("Unable to open " + args[0]);

        String ext = OEChem.OEGetFileExtension(args[1]);
        if (moldb.GetFormat() != OEChem.OEGetFileType(ext))
            OEChem.OEThrow.Fatal("Output format does not match input format: " +
                                 OEChem.OEGetFileExtension(args[0]) + " != " + ext);

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
            OEChem.OEThrow.Fatal("Unable to open " + args[1]);

        String title = args[2];
        for (uint idx = 0; idx < moldb.GetMaxMolIdx(); idx++)
        {
            if (title.Equals(moldb.GetTitle(idx)))
```

(continues on next page)

(continued from previous page)

```
                moldb.WriteMolecule(ofs, idx);
        }

    }
}
```

Note

Multi-conformer `.oeb` files can have multiple titles per molecule record. The top-level *OEMCMolBase* can have a title, as well as each *OEConfBase* can have a title. *OEMolDatabase.GetTitle* will only return the title of the top-level *OEMCMolBase* object and make no attempt to search for a title among the conformer data. In practice, this is fine since *OMEGA* will leave the *OEMCMolBase* title the same as the input file, and append warts to the individual conformer titles.

Index Files

The speed of *OEMolDatabase.Open* is limited by how fast data can be read from disk. For this reason, file position offsets can be precomputed and stored in a parallel `.idx` file. *OEMolDatabase.Open* will automatically detect the presence of this file based upon the file name of the file being opened and use those file offsets instead. For example, if the database file is called, "my_corporate_conformers.oeb", *OEMolDatabase.Open* will look for a file named "my_corporate_conformers.oeb.idx" to open as an index file. If an index file can not be located, a full file scan will occur instead. For files written once, and read many times, it can be highly beneficial to create a parallel index file with the *OECreateMolDatabaseIdx* function.

Note

OEMolDatabase.Save will automatically create a `.idx` file parallel to the file being saved. This behavior can be modified by the *OEMolDatabaseSaveOptions.SetWriteIdx* method on the *OEMolDatabaseSaveOptions* options class.

See also

- *OEGetMolDatabaseIdxFileName* for the default way index file names are created.

Database Generic Data

OEMolDatabase inherits from *OEBase*, allowing it to contain and round-trip generic data as described in the *Generic Data* chapter. This data is only written and read from `.oeb` files. It is stored in the `.oeb` file as a *OEHeader* record at the beginning of the file. *OEMolDatabase.Save* will write this record back out the `.oeb` file so that it be read by a subsequent *OEMolDatabase.Open* operation.

Caveats

Warning

OEMolDatabase will make an uncompressed copy of a molecular database file when opened on a `.gz`, GZipped file. The temporary file will be deleted upon the destruction of the *OEMolDatabase* object, however, it is still

recommended to not use `.gz` files with *OEMolDatabase* if it can be avoided. If a different directory is desired for the uncompressed file, alter the environment variables used by the `OEFileTempPath` function.

The largest caveat when working with *OEMolDatabase* objects is that they require a file on disk to provide storage. This allows the object to have a very small in-memory footprint at the expense of higher latency access to individual molecule records. The *OEMolDatabase* object makes no attempt to cache molecules that may be frequently read, instead leaving this up to the user at a higher level, or up to the operating system to cache frequently accessed disk pages.

The above reason is why `.gz` files are not well supported. The *OEMolDatabase* needs to be able to read a molecule record by seeking to a particular location in the file. Though the result is that multi-threaded applications can efficiently call `const` *OEMolDatabase* methods like `OEMolDatabase.GetMolecule` without any synchronization overhead.

2.1.22 Design Decisions

This chapter seeks to give a general overview of the major design decisions behind **OEChem TK**. The reader should have an understanding of the basic **OEChem TK** objects and functions. The reader should also be familiar with the basics of object oriented programming.

Throughout this manual, the *OEMol*, *OEGraphMol*, and *OEQMol* are the concrete classes which handle most molecular representation in **OEChem TK**. The *OEGraphMol* API is defined in the *OEMolBase* abstract base-class, the *OEMol* API is defined by the *OEMolBase* and `OEMCMolBase` abstract base-classes, and the *OEQMol* API is defined by the *OEMolBase* and *OEQMolBase* abstract base-classes. The `OEMCMolBase` class publicly inherits from the *OEMolBase* class, which in turn inherits from the *OEBase* class. Similarly, the *OEQMolBase* class publicly inherits from the *OEMolBase* class, which again inherits from the *OEBase* class. An *OEGraphMol*, can be passed to any function which takes an *OEMolBase* argument. An *OEMol* can be passed to any function which takes an `OEMCMolBase` or *OEMolBase* argument, and an *OEQMol* can be passed to any function which takes an *OEQMolBase* or *OEMolBase* argument. *Figure: Hierarchy A* represents the **OEChem TK** molecule hierarchy described in this paragraph.

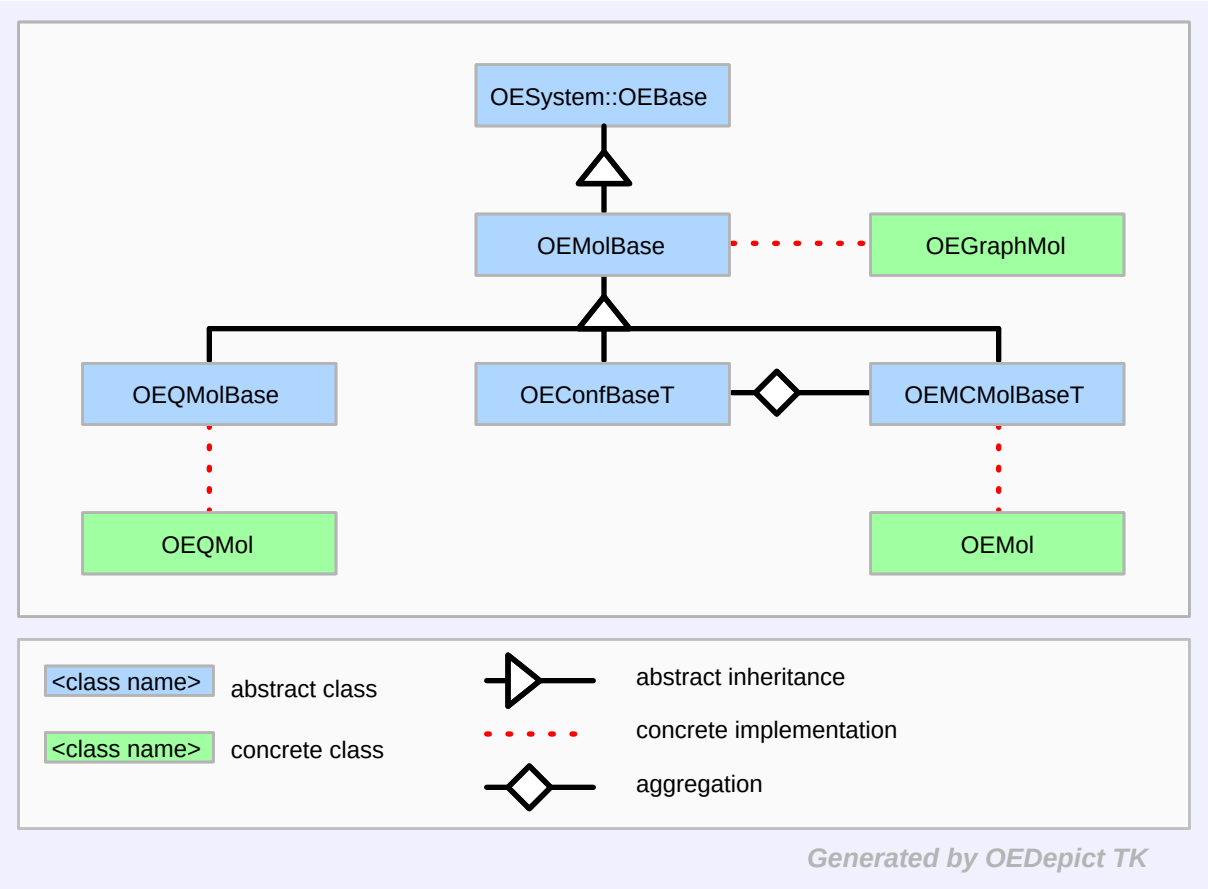


Fig. 62: **Hierarchy A**
OEChem TK Molecule Hierarchy

Abstract Base-Classes

The most important data types in the **OEChem TK** library are *OEMolBase*, *OEAtomBase*, and *OEBondBase*. These three classes describe the behaviors of molecules, atoms, and bonds respectively. However, these types are abstract classes, describing the methods and semantics of molecules, atoms and bonds, but without defining an actual implementation. (See *Figure: Hierarchy B*)

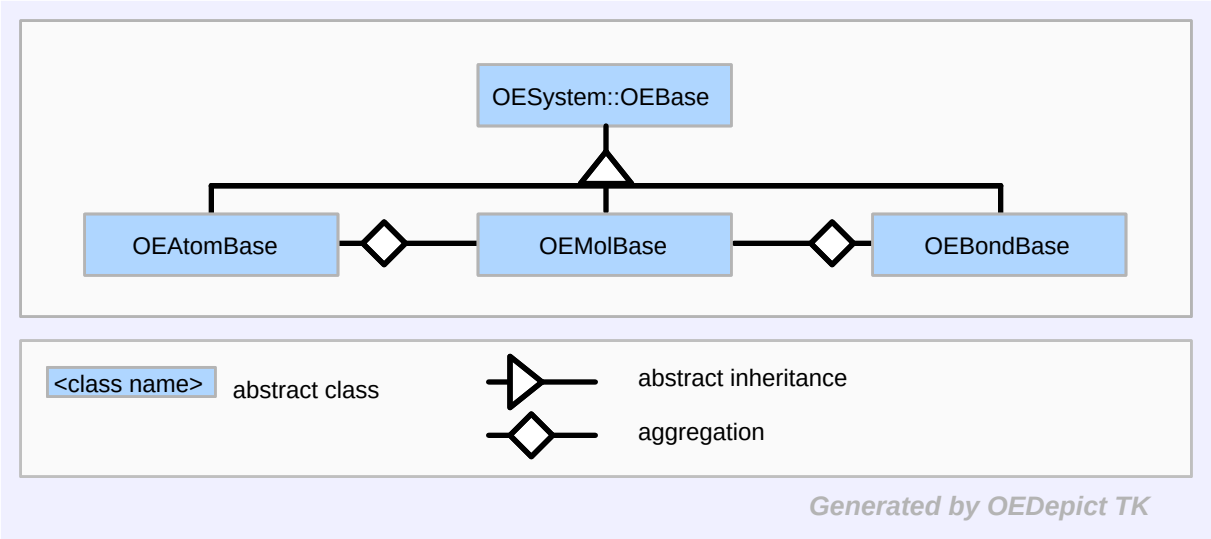


Fig. 63: **Hierarchy B**
OEChem TK Abstract Base-Classes

Hiding the implementation of these types is very important to the longevity of **OEChem TK**. It avoids the problem of the leaky abstraction. If code was written that was expecting a certain behavior from the implementation it would be fragile to arbitrary changes in implementation. Therefore, these abstract base-classes are considered stable, with a guarantee that they will not change from version to version.

Note

Since *OEAtomBase*, *OEBondBase*, and `OEConfBase` can only be accessed through their parent molecules, there is no need for concrete instances of these classes. In **OEChem TK**, these three classes are accessed via references to their respective base-classes, or through the iterator interface discussed in the *Atom and Bond Traversal* chapter.

Factory Functions

It is standard practice when working with abstract base-classes, such as the *OEMolBase*, to define a function which returns a pointer to one of these objects. These functions, called factories, give library users access to concrete objects even when only the abstract base-class is exposed in the public API. The following factory functions are provided to create molecule objects in **OEChem TK**. The associated symbolic namespace is used to control what type of molecule implementation is returned.

Factory Function	Symbolic Namespace
`OENewMolBase`	`OEMolBaseType`
`OENewMCMolBase`	`OEMCMolType`
`OENewQMolBase`	`OEQMolType`

Smart Pointers

The problem with factory functions is that they require the user to manage the object's memory. When the factory function returns a pointer to an abstract base-class, it also passes ownership of the memory to the programmer. To alleviate the problems associated with memory management introduced by factories, the smart-pointer idiom is often used. Simply put, a smart-pointer holds a real pointer to an object, and deallocates the pointer's memory when the smart-pointer goes out of scope (*e.g.* in its destructor). In **OEChem TK**, `OEGraphMols`, `OEMols`, and `OEQMols` fulfill

the function of both factories and smart-pointers. This gives the user access to multiple implementations without the need of worrying about memory management. The constructors allow the user to specify which implementation they would like using the associated symbolic namespace described in the previous table. Then the objects themselves act as smart-pointers, cleaning up the implementation pointers when the molecules go out of scope.

Listing 1 demonstrates how to specify an alternative molecule implementation. A *SMILES* string is then read into different implementation. This demonstrates how implementations that adhere to the *OEMolBase* interface can reuse existing algorithms written towards it.

Listing 1: Using an alternative molecule implementation

```
using System;
using OpenEye.OEChem;

public class MolBaseType
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol(OEMolBaseType.OEMiniMol);

        OEChem.OESmilesToMol(mol, "C1=CC=CC=C1");

        string isosmi = OEChem.OEMolToSmiles(mol);
        Console.WriteLine("Canonical isomeric SMILES is " + isosmi);
        return 0;
    }
}
```

Query Molecules

It is not uncommon in chemical informatics to consider the equivalence of the graph which represents a molecule and the graph which represents a substructure query. Indeed the simplest of queries are molecules themselves. If one considers each node (or atom) as a potentially complex atom query, even highly complex queries can be represented as molecules.

In **OEChem TK**, this concept of a query as a molecule is represented by the *OEQMolBase* abstract base-class. An *OEQMolBase* contains *OEQAtomBase* and *OEQBondBase* atom and bond representations. These versions of **OEChem TK** atoms and bonds manage the atom and bond expressions which comprise the query.

OEChem TK defines the concrete *OEQMol* class which provides a programmer access to the API defined by the *OEQMolBase* class. This is analogous to the way an *OEGraphMol* provides concrete access to *OEMolBase* API. *Figure: Hierarchy C* represents the **OEChem TK** query molecule hierarchy.

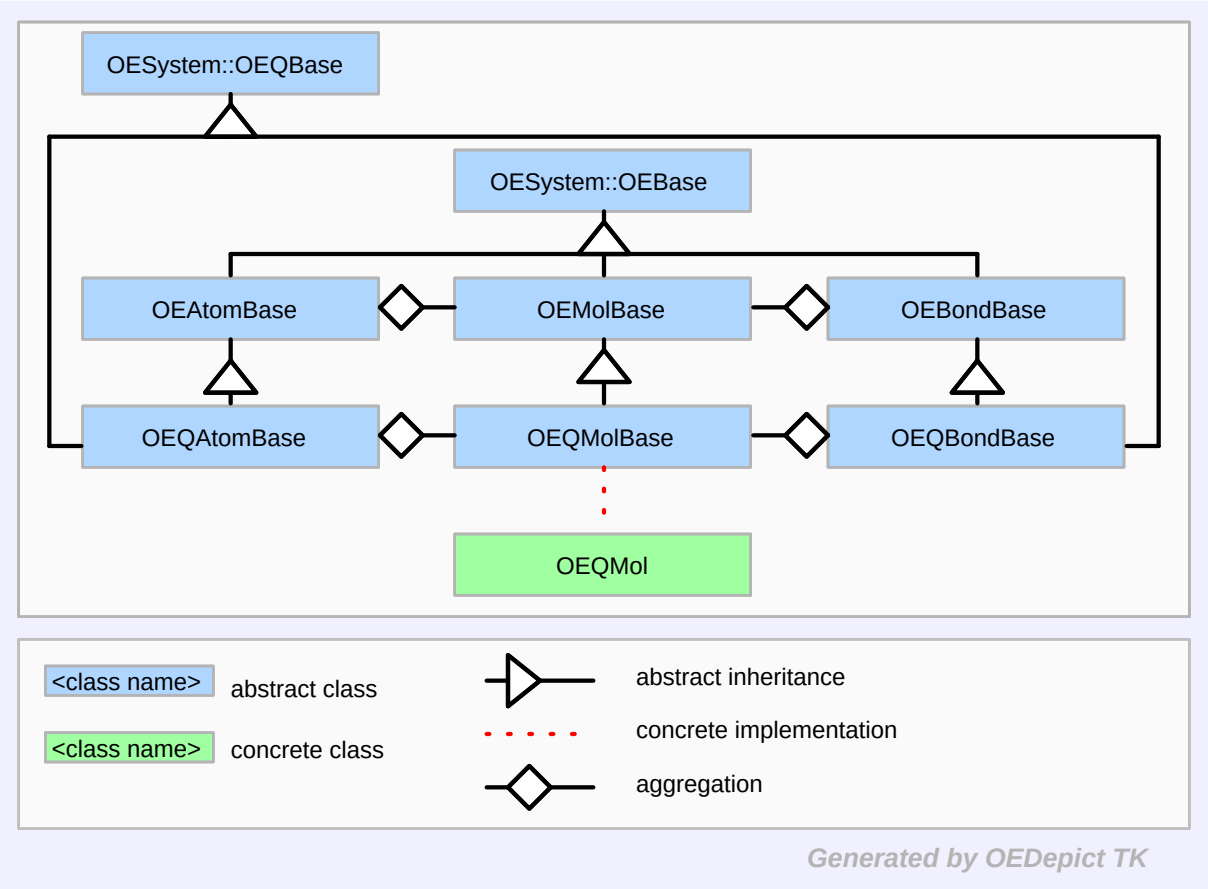


Fig. 64: **Hierarchy C**
OEChem TK Query Molecule Hierarchy

This abstraction allows **OEChem TK** to treat separate query mechanisms to work similarly. This is similar to how **OEChem TK** can represent any molecular file format through a single *OEMolBase* interface. For example, *SMARTS* can be parsed into an *OEQMolBase* using the `OEParseSmarts` function. Also, *MDL Queries* can be parsed into an *OEQMolBase* using the `OEReadMDLQueryFile` file.

See also

Pattern Matching for a general discussion of **OEChem TK** pattern matching facilities.

Free-Functions

OEChem TK is an object-oriented library. However, the design philosophy is that molecule objects are primarily data containers with data access member functions. Most powerful data analysis and manipulation routines in **OEChem TK** are implemented as free functions rather than member functions. This decision is based on the realization that the abstraction of a molecule can be neither stable nor consistent. To one programmer, a molecule should describe and perceive the space group of organometallic complexes, while to another a molecule should describe the residues and secondary structure of a macromolecule. Both of these perspectives are reasonable and should be supported. Further, a nearly infinite additional list of molecule designs can be imagined and should be supported. In order to do this, the **OEChem TK** molecule must be extensible, light-weight, and easily re-implemented. Thus, major changes to the **OEChem TK** molecules can be made, without need to re-implement more than a handful of functions. Conversely, an entire new area of chemistry can be added to the **OEChem TK** repertoire through free-functions without needing to

implement the function in multiple molecules. We have used namespaces extensively to keep the free-functions from cluttering the global namespace.

The concrete classes are provided for the reasons described in the *Smart Pointers* section. Function parameters should never specify these concrete classes. Instead, they should always be declared using the corresponding abstract base-classes. For instance, rather than:

```
void fcn(OEGraphMol mol)
{
  ...
}
```

Functions parameters should use the appropriate abstract base-class like the following:

```
void fcn(OEMolBase mol)
{
  ...
}
```

See also

The *OEChem Functions* API section for a complete listing of all of **OEChem TK**'s free-functions.

Programming Layers: The Deep and Twisted Path

OEChem TK was designed to provide a library which puts powerful algorithms in the hands of novice users without hand-cuffing the expert. For this reason, **OEChem TK** can at the same time seem trivial and overwhelming. There are often several ways to carry out certain tasks in **OEChem TK** each with its subtle advantages, which can benefit an experienced user. There are very few algorithms we have shied away from including in **OEChem TK**, and many of the methods are new, unique and powerful. This gives **OEChem TK** a very rich interface, yet to gain this efficiency and power **OEChem TK** may force the user to think about problems in ways they are not accustomed to doing. The hope is that the user can benefit from the experience.

OEChem TK has several layers of interfaces to most of its functionality. There are "high-level" interfaces, which provide the user with an enormous amount of power with minimal code. This level is exemplified by the `OEReadMolecule` and `OEWriteMolecule` functions. With these functions the functionality of the `babel` file-format conversion program is trivial. While this is trivial to write and understand (maybe after understanding this manual), it should not belie the fact that **OEChem TK** is carrying out an enormous amount of work under the surface.

A perhaps "mid-level" interface in **OEChem TK** is the ability to fine tune molecule file-formats using flavors described in *Flavored Input and Output* section. Flavors are most useful when file-formats are being used for more than they were originally intended, for example, PDB files. While this functionality is perhaps not for the first-day user, it certainly doesn't require a stout heart.

Finally, for advanced programmers, **OEChem TK** provides access to nearly all of the details. **OEChem TK** molecules have a simple API which can be used to derived custom molecule implementations. The free-function heavy API lets the user apply **OEChem TK's** powerful algorithms on custom data structures. Similarly, many of the functions that are wrapped in high level functions (like the molecule readers and writers) are also available directly to the user at the low level. For instance, `OEWritePDBFile` allows the user to write out a PDB file with a very specific flavor without any normalization that `OEWriteMolecule` may perform.

OEChem TK is a still a live moving software project. If a roadblock is found a particular level, dig deeper into the next level. Often, the functionality is already present.

2.1.23 Multi-Threading

OpenEye deems only the following toolkits "thread-safe":

- OEChem TK (including oeplatform, oesystem, oechem, and oebio)
- OEDepict TK
- OEDocking TK
- Grapheme TK
- GraphSim TK
- Lexichem TK
- Omega TK

> **Warning**
>
> Absolutely no guarantee is made about the thread-safety of any other library not on the preceding list.

The most common reason to multi-thread a program is for performance. However, threading has certain overhead costs associated with it. The user should profile their application to determine the best possible way to parallelize it. There is no guarantee that multi-threading a program will make it run faster, in fact, it can make it run dramatically more slowly. Some of the examples presented in this chapter will perform more slowly than their single-threaded counterparts. They were chosen for the simplicity of demonstrating certain concepts. The user should perform timing analysis on the code in this chapter on their own systems to determine whether the methods are appropriate for them.

Input and Output Threads

OEChem TK provides a rather advanced mechanism for spawning a thread to handle I/O asynchronously. Parallelizing I/O operations can yield significant performance gains when dealing with bulky file formats like SDF and MOL2, or when dealing with files on network mounted directories. Naturally, performance gains may vary based upon the system configuration in question.

The input and output threads are encapsulated inside the *oemolthread* abstraction. *oemolthreads* are designed to behave as similarly as possible to *oemolstreams*. This should allow easy drop in replacement to existing code by simply changing the type of a variable from *oemolistream* to *oemolithread* for input, and *oemolostream* to *oemolothread* for output.

For example, the *MolFind* program described in the *Substructure Search* section can be modified to the nearly identical program in *Listing 1*.

Listing 1: Using Threaded Input and Output

```
using System;
using OpenEye.OEChem;

public class ThreadedMolFind
{
    public static void Main(string[] argv)
    {
        OESubSearch ss = new OESubSearch(argv[0]);

        oemolithread ifs = new oemolithread(argv[1]);
        oemolothread ofs = new oemolothread(argv[2]);
```

(continues on next page)

(continued from previous page)

```
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEPrepareSearch(mol, ss);
            if (ss.SingleMatch(mol))
            {
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }
    }
}
```

Note

Both *oemolstreams* and *oemolthreads* should be evaluated for performance. *oemolthreads* increase user time CPU usage to save on potentially expensive system calls.

The difference in *Listing 1* may be a little difficult to spot for the experienced **OEChem TK** programmer used to seeing *oemolstream* declarations, however, the following two declarations are not *oemolstreams*. They are *oemolthreads* meant to behave as closely as possible to *oemolstreams*.

```
        oemolithread ifs = new oemolithread(argv[1]);
        oemolothread ofs = new oemolothread(argv[2]);
```

However, *oemolthreads* could not be made to behave *exactly* like *oemolstreams*. The following key differences between *oemolthreads* and *oemolstreams* should be taken into account:

oemolthreads do not support stream methods: *read*, *seek*, *size*, and *tell*.

> These do not make much sense in a multi-threaded environment where the I/O thread can be at any arbitrary location in the file. Using the methods would be prone to race conditions.

The *oemolithread* default *constructor* does not default to `stdin`.

> The default constructor of *oemolistream* will open the stream on `stdin`. However, read operations are handled by a separate opaque I/O thread inside the *oemolithread*. The I/O thread uses blocking reads that do not return unless the user closes `stdin` of the process. *oemolithreads* can still read from `stdin` using the same syntactic trick described in the *Command Line Format Control* section supplied to *open* or the *constructor* which takes a `string`.

The *oemolithread* does not support the *SetConfTest* interface.

> As will be described later, the I/O threads inside *oemolthreads* do a minimal amount of work to maximize scalability in parallel applications. The work described in the *Input and Output* section to recover multi-conformer molecules from single-conformer file formats can be quite expensive. Note, this does not preclude *oemolithreads* from reading multi-conformer molecules from file formats that support them inherently, *e.g.*, multi-conformer molecules can be read out of multi-conformer OEB files through *oemolithreads*.

With all the preceding considerations taken into account the real gain of *oemolthreads* is that *OEReadMolecule* and *OEWriteMolecule* can be safely called from multiple threads onto the same *oemolthread*. They are guaranteed to be free of race conditions when applied to *oemolthreads*. Not only that, the expensive parsing (*OEReadMolecule*) and serialization (*OEWriteMolecule*) occurs in the thread that calls the function, not in the *oemolthread* itself. The *oemolthread* takes care of I/O and compression if reading or writing to a `.gz` file.

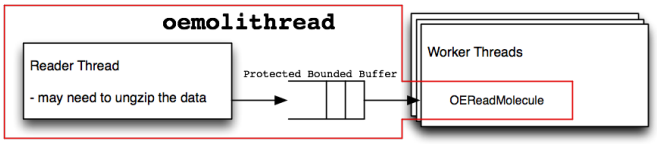


Fig. 65: **oemolithread**
Diagram of data flow inside an oemolithread

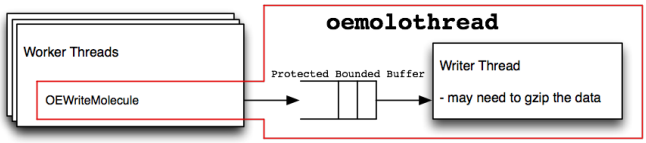


Fig. 66: **oemolothread**
Diagram of data flow inside an oemolothread

Figure: oemolithread and *Figure: oemolothread* shows how data flows between an arbitrary number of worker threads and the I/O thread. Molecule data passes through a protected bounded buffer. It is "protected" against race conditions when multiple threads operate on the buffer. It is "bounded" in size to prevent increasing the required amount of memory by inserting many items into the buffer faster than can be consumed on the other side. The I/O threads will be put to sleep, allowing other threads to use the CPU, when there is not a sufficient amount of work for them to do. This means the number of worker threads should typically be the number of available CPUs. The I/O threads don't require much CPU time, therefore, they don't need their own dedicated CPU. This could change as the number of worker threads increases, or if the I/O thread is expected to handle data compression or uncompression.

Listing 2 demonstrates how to scale the *molecule searching* program to multiple CPUs. A worker thread is dedicated for each CPU. There are also I/O threads to handle the input and output operations. The data parsing (*OEReadMolecule*), substructure searching(*OESubSearch.SingleMatch*), and data serialization (*OEWriteMolecule*) is all done in parallel by the worker threads. The costly synchronization point of reading and writing data is handled by the separate I/O threads.

Listing 2: Using multiple threads to read and write to molthreads

```
using System;
using System.Collections.Generic;
using System.Threading;
using OpenEye.OEChem;

public class MultiCoreMolFind
{
    private static OESubSearch ss;
    private static oemolithread ifs;
    private static oemolothread ofs;

    public static int Main(string[] args)
    {
        ss  = new OESubSearch(args[0]);
        ifs = new oemolithread(args[1]);
        ofs = new oemolothread(args[2]);

        int ncpus = Environment.ProcessorCount;
```

(continues on next page)

(continued from previous page)

```
        List<Thread> thrds = new List<Thread>();

        for (int i = 0; i < ncpus; i++)
        {
            MolFindThread mft = new MolFindThread();
            ThreadStart job = new ThreadStart(mft.Run);
            Thread thrd = new Thread(job);

            try
            {
                thrd.Start();
                thrds.Add(thrd);
            }
            catch (Exception)
            {
                Console.WriteLine("Exception caught");
            }
        }

        foreach (Thread thrd in thrds)
        {
            thrd.Join();
        }

        return 0;
    }

    public class MolFindThread
    {
        public void Run()
        {
            OEGraphMol mol = new OEGraphMol();
            while (OEChem.OEReadMolecule(ifs, mol))
            {
                OEChem.OEPrepareSearch(mol, ss);
                if (ss.SingleMatch(mol))
                {
                    OEChem.OEWriteMolecule(ofs, mol);
                }
            }
        }
    }
}
```

Listing 2 will generate the same set of molecules in the output file as *Listing 1*. However, there is big difference between the output files, the molecule order is non-deterministic. For most users this does not matter, database files are treated as arbitrarily ordered sets of molecules. If the order of the molecule database is important do not use the program in *Listing 2*.

Note

Database order is preserved by the program in *Listing 1* because there is only one worker thread. This allows users with database order restrictions to still take advantage of threaded asynchronous I/O.

Thread Safety

OEChem TK is considered to be a "thread-safe" library. It makes the same guarantees the C++ Standard Template Library (STL) makes in concern of threads:

```
Unless otherwise specified, standard library classes may safely be
instantiated from multiple threads and standard library functions
are reentrant, but non-const use of objects of standard library
types is not safe if shared between threads. Use of non-constness
to determine thread-safety requirements ensures consistent
thread-safety specifications without having to add additional
wording to each and every standard library type.
```

See also

- Thread-Safety in the Standard Library

Classes

Unless otherwise stated in the API documentation, *OEChem objects* can be instantiated and used in multiple threads. For example, in *Listing 2*, special care was taken to make the molecule object local to each thread.

This was accomplished by making an *OEGraphMol* instance local to the thread's `run` method. Therefore, ensuring each thread had a separate *OEGraphMol* to read molecules into.

```
public void Run()
{
    OEGraphMol mol = new OEGraphMol();
    while (OEChem.OEReadMolecule(ifs, mol))
    {
```

The same object can be accessed from multiple threads if all access is through a `const` method. A `const` method is any method that does not change data internal to the class. In C++, a method can be explicitly marked `const`, allowing the compiler to cause the code to not compile if this guarantee is not kept.

For example, the *OESubSearch.SingleMatch* method is a `const` method. Therefore, the same *OESubSearch* object can be used from multiple threads without the need to make local copies of the *OESubSearch* object in *Listing 2*. However, notice that the molecule argument passed to *OESubSearch.SingleMatch* is local to each thread. Otherwise, one thread could be reading data into the molecule object while another thread is trying to perform the substructure search, leading to a race condition and undefined behavior.

```
if (ss.SingleMatch(mol))
{
```

Warning

Care must be taken when reusing the same objects in multiple threads. Only one invocation of a non-`const` method (*e.g.* *`OESubSearch.AddConstraint`*) concurrently could crash the program. Concurrent execution of non-`const` methods must to be protected by *`mutual exclusion`*.

Functions

OEChem functions, unless otherwise stated in the API documentation, can be safely called from multiple threads. For example, *`OEGetTag`* needs to access a global table to make `string` to `unsigned integer` associations global to the program. *`OEGetTag`* ensures its own thread safety so that it is safe to call from multiple threads.

However, mutual exclusion can be very expensive. This is why **OEChem TK** (and the STL) make no effort to make concurrent object access safe. So when mutual exclusion can not be avoided, as in the case of *`OEGetTag`*, faster alternatives should be sought. For example, the return of *`OEGetTag`* can be cached to a local variable.

It is important to realize that even though the function may be thread-safe, the function may be altering the state of its arguments, which could be shared across threads. Again, the "const-ness" of the arguments must be taken into account. For example, it is not safe to call *`OEWriteMolecule`* on the same molecule from separate threads (even on different *`oemolostreams`*) because the molecule argument to *`OEWriteMolecule`* is non-`const`. The molecule argument is non-`const` for performance, the molecule may need to be changed to conform to the file format being written to. **OEChem TK** provides *`OEWriteConstMolecule`* that will make a local copy of the molecule before making necessary changes for the file format being requested.

Exceptions to the `const` argument rule must be explicitly stated in the API documentation. For example, the *`OEWriteMolecule`* function applied to the non-`const` argument of *oemolothread* is thread-safe only by virtue of design.

```
OEChem.OEWriteMolecule(ofs, mol);
```

Memory Management

Memory allocation is often a bottleneck in any program. This is especially the case in multi-threaded programs because memory allocation must protect a globally shared resource, the memory space of the process. Most `malloc` implementations use mutual exclusion to make themselves thread-safe. However, mutual exclusion is expensive, and cuts down on the scalability of the multi-threaded program. **OEChem TK** attempts to alleviate the problem by implementing a memory cache for the following small objects:

- Any **OpenEye Toolkit** *iterator object*, i.e., something derived from *OEIterBase*.
- Any *OEGetNbrAtom* functor. This is used by some implementations of *`OEAtomBase.GetAtoms`* that returns an iterator as well, see previous.
- The *OEMatch* class. Used by almost any API that operates with an instance of *OEMatchBase*.
- *`OEMolBaseType.OEMiniMol`* instances of *OEGraphMol*.

The cache is totally opaque to the normal **OEChem TK** user. However, it can cause problems in multi-threaded situations.

The small object cache implementation is controlled through the *`OESetMemPoolMode`* function. The parameter passed to this function determines what sort of cache implementation to use, the possible options are a set of values bit-wise or'd from the *`OEMemPoolMode`* constant namespace. The possible combinations are as follows:

SingleThreaded|BoundedCache
: Makes no attempt at thread safety. Will only cache allocations up to a certain size, then further allocations will be sent to the global `operator new(size_t)` function.

SingleThreaded|UnboundedCache
Makes no attempt at thread safety. Will cache all allocations up to any size, never returning memory to the operating system after it has been requested.

Mutexed|BoundedCache
Protects a single global cache with a mutex. Will only cache allocations up to a certain size, then further allocations will be sent to the global `operator new(size_t)` function.

Mutexed|UnboundedCache
Protects a single global cache with a mutex. Will cache all allocations up to any size, never returning memory to the operating system after it has been requested.

ThreadLocal|BoundedCache
Uses a separate cache for each thread. Will only cache allocations up to a certain size, then further allocations will be sent to the global `operator new(size_t)` function.

ThreadLocal|UnboundedCache
Uses a separate cache for each thread. Will cache all allocations up to any size, never returning memory to the operating system after it has been requested.

System
Forwards all allocations to the global `operator new(size_t)` function. This allows the user to circumvent the entire caching system in preference to their own `operator new(size_t)` function allowing other third party memory allocators to be linked against.

The default implementation in C# is `Mutexed|UnboundedCache`. *Unbounded* for performance, however, *Mutexed* since the C# garbage collector runs in a separate thread.

Thread Safety Options

If database order is a necessity it may be possible to gain performance from threading using "pipeline" parallelism. *Listing 3* demonstrates how the molecule searching problem can be broken down into 5 stages, each running in a separate thread. In order, the 5 stages are:

1. Reading data from disk - *oemolithread*
2. Parsing the data into a molecule - `ParseThread`
3. Performing the substructure search on the molecule - `SearchThread`
4. Serializing the molecule - `main thread`
5. Writing data to disk - *oemolothread*

Listing 3: Using pipeline parallelism in Java OEChem TK

```
using System;
using System.Collections.Generic;
using System.Threading;
using OpenEye.OEChem;

public class PipelineMolFind
{
    private static OESubSearch ss;
    private static oemolithread ifs;
    private static oemolothread ofs;

    private static BlockingQueue<OEGraphMol> iqueue;
    private static BlockingQueue<OEGraphMol> oqueue;
```

(continues on next page)

(continued from previous page)

```
    public static int Main(string[] args)
    {
        ss   = new OESubSearch(args[0]);
        ifs = new oemolithread(args[1]);
        ofs = new oemolothread(args[2]);

        iqueue = new BlockingQueue<OEGraphMol>(1024);
        oqueue = new BlockingQueue<OEGraphMol>(1024);

        ParseThread pthrdInstance = new ParseThread();
        Thread pthrd = new Thread(new ThreadStart(pthrdInstance.Run));
        pthrd.Start();

        SearchThread sthrdInstance = new SearchThread();
        Thread sthrd = new Thread(new ThreadStart(sthrdInstance.Run));
        sthrd.Start();

        try
        {
            OEGraphMol mol = oqueue.Take();
            while (mol.IsValid())
            {
                OEChem.OEWriteMolecule(ofs, mol);
                mol = oqueue.Take();
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Exception thrown!!");
        }

        ofs.close();
        return 0;
    }

    public class BlockingQueue<T>
    {
        private Queue<T> q;

        public BlockingQueue()
        {
            this.q = new Queue<T>();
        }

        public BlockingQueue(int capacity)
        {
            this.q = new Queue<T>(capacity);
        }

        public void Put(T element)
        {
```

(continues on next page)

(continued from previous page)

```
            this.q.Enqueue(element);
            lock (this.q)
            {
                Monitor.Pulse(this.q);
            }
        }

        public T Take()
        {
            lock (this.q)
            {
                while (this.q.Count == 0)
                {
                    Monitor.Wait(this.q);
                }

                return this.q.Dequeue();
            }
        }
    }

    public class ParseThread
    {
        public void Run()
        {
            try
            {
                OEGraphMol mol = new OEGraphMol();
                while (OEChem.OEReadMolecule(ifs, mol))
                {
                    iqueue.Put(mol);
                    mol = new OEGraphMol();
                }

                mol.Clear();
                // signal SearchThread to die with an empty molecule
                iqueue.Put(mol);
            }
            catch (Exception)
            {
                Console.WriteLine("Exception thrown!!");
            }
        }
    }

    public class SearchThread
    {
        public void Run()
        {
            try
            {
                OEGraphMol mol = iqueue.Take();
```

(continues on next page)

(continued from previous page)

```
                while (mol.IsValid())
                {
                    OEChem.OEPrepareSearch(mol, ss);
                    if (ss.SingleMatch(mol))
                    {
                        oqueue.Put(mol);
                    }

                    mol = iqueue.Take();
                }

                mol = new OEGraphMol();
                // signal main thread to die with an empty molecule
                oqueue.Put(mol);
            }
            catch (Exception)
            {
                Console.WriteLine("Exception Thrown!!");
            }
        }
    }
}
```

Overriding the allocator

OpenEye Toolkits are hardcoded to use *OEMemPoolMode.System* for C# and can not be changed at this time. Please contact support@eyesopen.com if you think this option may help you.

(Un)bounded Options

Two types of small object caches can be set with *OESetMemPoolMode*: *Bounded* and *Unbounded*. The default is an *Unbounded* cache for performance. The assumption is most **OEChem TK** programs will have a constant memory usage. Even if the program has a steadily rising memory footprint the program will usually exit after it has maxed out its memory requirements.

This may not be the case for long running programs, for example, a web service. These programs may need to create a temporarily large memory footprint and then release it back to the system for some other program to use. An *Unbounded* cache will hold onto all memory it sees, never returning any to the operating system.

A *Bounded* cache will only cache allocations up to a limit. After that limit has been reached all further allocations will be sent to the system memory allocator. This allows the deallocations later in the program to return the memory to the operating system. This allows certain objects to still use the fast cache while the other objects that are bloating the memory footprint will incur the cost of asking the operating system for the memory they require.

2.1.24 Predicate Functors

Functors that return bool are an important special case. A unary function whose return type is bool is called a *predicate*.

In **OEChem TK**, these functors are often passed into another function. The functors are then called from inside the second function. This is the concept of a *callback*, because the second function provides the argument and 'call's back' to the functor which was passed into the function. Generator method such as *OEMolBase.GetAtoms* can take a functor as an argument and use the callback mechanism to iterate over atoms that satisfy the functor passed to it. See example in *Atom or Bond Subset Iteration* section.

In the example below, the function `CountAtoms` loops over the atoms and performs a call-back to the predicate functor `pred` for each atom. If the predicate returns true, a counter is incremented. The `main` function passes *OEIsOxygen* predefined atom predicates to the `CountAtoms` function that counts the number of oxygen atoms in the molecule. (Please note that this function is already implemented in **OEChem TK** and called *OECount*.)

Listing 1: Using functor callbacks

```
using System;
using System.IO;
using OpenEye.OEChem;

public class Functors
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cc[nH]c1CC2COCNC2");
        Console.WriteLine("Number of oxygen atoms = " + CountAtoms(mol, new␣
↪OEIsOxygen()));
    }

    private static int CountAtoms(OEMolBase mol, OEUnaryAtomPred pred)
    {
        int counts = 0;

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            if (pred.Call(atom))
            {
                counts++;
            }
        }

        return counts;
    }
}
```

Built-in Functors

There are many useful functors already defined in **OEChem TK**. These can be used by programmers with little or no understanding of the details of how functors work. A programmer can simply pass them to one of the many **OEChem TK** functions and methods which take predicates as arguments.

Atom Functors

Access	Functor Name
ring atoms	*OEAtomIsInRing*
chain atoms	*OEAtomIsInChain*
atom with specified atom index	*OEHasAtomIdx*
atom with selected atom index	*OEAtomIdxSelected*
atom with specified atom name	*OEHasAtomName*
atoms with specified atom stereo	*OEHasAtomStereoSpecified*
atoms with specified formal charge	*OEHasFormalCharge*
atoms with specified number of heavy atom neighbors	*OEHasHvyDegree*
aromatic atoms	*OEIsAromaticAtom*
atoms with specific hybridization	*OEIsAtomHybridization*
chiral atoms	*OEIsChiralAtom*
atoms with anisotropic B-factor parameters	*OEHasAnisou*
atoms with specified map index	*OEHasMapIdx*
atoms representing R-Groups	*OEIsRGroup*
valid atoms (by OpenEye valence conventions)	*OEIsValidAtomValence*
valid atoms (by MDL valence conventions)	*OEIsValidMDLAtomValence*
n^{th} atom	*OENthAtom*
atoms that are both terminal and heavy	*OEIsTermHeavyAtom*
atom membership in a set of atoms	OEIsAtomMember

Listing 2: Using predefined atom functors

```
using System;
using OpenEye.OEChem;

public class FunctorsAtom
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cc[nH]c1CC2COCNC2");
        Console.WriteLine("Number of heavy atoms = " +
                          OEChem.OECount(mol, new OEIsHeavy()));
        Console.WriteLine("Number of ring atoms  = " +
                          OEChem.OECount(mol, new OEAtomIsInRing()));
    }
}
```

The output of the preceding program is the following:

```
Number of heavy atoms = 12
Number of ring atoms  = 11
```

Atomic Number Functors

Access	Functor Name
atoms with specified atomic number	*OEHasAtomicNum*
carbon atoms	*OEIsCarbon*
halogen atoms	*OEIsHalogen*
heavy atoms	*OEIsHeavy*
hetero atoms	*OEIsHetero*
explicit hydrogen atoms	*OEIsHydrogen*
metal atoms	*OEIsMetal*
nitrogen atoms	*OEIsNitrogen*
oxygen atoms	*OEIsOxygen*
sulfur atoms	*OEIsSulfur*
phosphorus atoms	*OEIsPhosphorus*
non-carbon atoms	*OEIsPolar*
polar hydrogen atoms	*OEIsPolarHydrogen*

Please note that the following two lines produce the same result.

```
        Console.WriteLine("Number of oxygen atoms = " +
                          OEChem.OECount(mol, new OEHasAtomicNum(OEElemNo.O)));
        Console.WriteLine("Number of oxygen atoms = " +
                          OEChem.OECount(mol, new OEIsOxygen()));
```

Bond Functors

Access	Functor Name
ring bonds	*OEBondIsInRing*
chain bonds	*OEBondIsInChain*
bond with specified bond index	*OEHasBondIdx*
bond with selected bond index	*OEBondIdxSelected*
bonds with specified bond order	*OEHasOrder*
rotatable bonds	*OEIsRotor*
chiral bonds	*OEIsChiralBond*
bonds with specific bond stereo	*OEHasBondStereoSpecified*
aromatic bonds	*OEIsAromaticBond*
bond membership in a set of bonds	OEIsBondMember

Listing 3: Using predefined bond functors

```
using System;
using OpenEye.OEChem;

public class FunctorsBond
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CC(=O)Nc1c[nH]cc1");
```

(continues on next page)

(continued from previous page)

```
        Console.WriteLine("Number of ring bonds  = " +
                          OEChem.OECount(mol, new OEBondIsInRing()));
        Console.WriteLine("Number of rotor bonds = " +
                          OEChem.OECount(mol, new OEIsRotor()));
    }
}
```

The output of the preceding program is the following:

```
Number of ring bonds  = 5
Number of rotor bonds = 2
```

Group Functors

Access	Functor Name
groups with a specific atom	*OEHasAtomInGroup*
groups with a specific bond	*OEHasBondInGroup*
groups with a specific type	*OEHasGroupType*
groups that store MDL stereo information	*OEIsMDLStereoGroup*

Reaction Component Functors

Access	Functor Name
atoms of the catalysts or solvents of a reaction	*OEAtomIsInAgent*
atoms of the product molecule(s)	*OEAtomIsInProduct*
atoms of the reactant molecule(s)	*OEAtomIsInReactant*

Conformer Functors

Access	Functor Name
conformer with specified index	*OEHasConfIdx*
conformer with selected index	*OEConfIdxSelected*

Residue Data Functors

Access	Functor Name
atoms with specified residue properties	*OEAtomMatchResidue*
atoms with specified chain id	*OEHasChainID*
atoms with specified residue number	*OEHasResidueNumber*
atoms with an alternate location	*OEHasAlternateLocation*
atoms with specified fragment number	*OEHasFragmentNumber*
atoms with specified PDB index	*OEHasPDBAtomIndex*
alpha carbon	*OEIsCAlpha*
backbone atom	*OEIsBackboneAtom*
water	*OEIsWater*
nucleic acid base	*OEIsNucleicAcidBase*
nucleic acid sugar	*OEIsNucleicAcidSugar*
nucleic acid phosphate	*OEIsNucleicAcidPhosphate*

Composite Functors

Occasionally, one may want to use a logical operator to join two or more functors. The following table shows the composite functors defined in **OEChem TK**.

Table 13: **Composite Functors**

Logical Not	Logical or	Logical and	Description
OENotAtom	*OEOrAtom*	*OEAndAtom*	atom composite functors
OENotBond	*OEOrBond*	*OEAndBond*	bond composite functors
OENotConf	*OEOrConf*	*OEAndConf*	conformation composite functors
OENotGroup	*OEOrGroup*	*OEAndGroup*	group composite functors
OENotRoleSet	*OEOrRoleSet*	*OEAndRoleSet*	roleset composite functors

Each composite functor takes the appropriate number of predicates as arguments and generates a single unary predicate. The following example demonstrates how to use composite functors to build expressions from **OEChem TK**'s predefined atom predicates.

Listing 4: Combining predefined atom predicates

```
using System;
using OpenEye.OEChem;

public class CompositionFunctorsAtom
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cnc(O)cc1CCCBr");
        Console.WriteLine("Number of chain atoms = " +
                          OEChem.OECount(mol,
                                         new OENotAtom(new OEAtomIsInRing())));
        Console.WriteLine("Number of aromatic nitrogens = " +
                          OEChem.OECount(mol,
                                         new OEAndAtom(new OEIsNitrogen(), new␣
```

(continues on next page)

(continued from previous page)

```
↪OEIsAromaticAtom()));
        Console.WriteLine("Number of non-carbons = " +
                          OEChem.OECount(mol,
                                         new OENotAtom(new OEHasAtomicNum(OEElemNo.
↪C))));
        Console.WriteLine("Number of nitrogen and oxygen atoms = " +
                          OEChem.OECount(mol, new OEOrAtom(new OEHasAtomicNum(OEElemNo.
↪N), new OEHasAtomicNum(OEElemNo.O))));
    }
}
```

The *OECount* function returns the number or objects (in this case atoms) matching the given predicate argument.

The output of the preceding program is the following:

```
Number of chain atoms = 5
Number of aromatic nitrogens = 1
Number of non-carbons = 3
Number of nitrogen and oxygen atoms = 2
```

Composite functors can be used similarly to combine predefined bond predicates.

Listing 5: Combining predefined bond predicates

```
using System;
using OpenEye.OEChem;

public class CompositionFunctorsBond
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "N#CCC1CCNC=C1");
        Console.WriteLine("Number of non-rotatable bonds = " +
                          OEChem.OECount(mol,
                                         new OENotBond(new OEIsRotor())));
        Console.WriteLine("Number of ring double bonds = " +
                          OEChem.OECount(mol,
                                         new OEAndBond(new OEBondIsInRing(), new␣
↪OEHasOrder(2))));
        Console.WriteLine("Number of double or triple bonds = " +
                          OEChem.OECount(mol,
                                         new OEOrBond(new OEHasOrder(2), new␣
↪OEHasOrder(3))));
    }
}
```

The output of the preceding program is the following:

```
Number of non-rotatable bonds = 8
Number of ring double bonds = 1
Number of double or triple bonds = 2
```

User Defined Functors

While many predefined functors exist in **OEChem TK**, it is not difficult to find a situation which calls for a new user-defined functor.

User-defined functor can be written by deriving from either the *OEUnaryAtomPred* or the *OEUnaryBondPred* class.

The following example shows a user defined atom functor which returns true for aliphatic nitrogens.

Listing 6: User defined atom predicate

```
using System;
using OpenEye.OEChem;

public class UserPredAtom
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cc[nH]c1CC2COCNC2");
        Console.WriteLine("Number of aliphatic N atoms = " +
                          OEChem.OECount(mol, new PredAliphaticNitrogen()));
    }

    private class PredAliphaticNitrogen : OEUnaryAtomPred
    {
        public override bool Call(OEAtomBase atom)
        {
            return atom.IsNitrogen() && !atom.IsAromatic();
        }

        public override OEUnaryAtomBoolFunc CreateCopy()
        {
            OEUnaryAtomBoolFunc copy = new PredAliphaticNitrogen();
            copy.ReleaseOwnership();
            return copy;
        }
    }
}
```

The output of the preceding program is the following:

```
Number of aliphatic N atoms = 1
```

A bond predicate can be similarly defined by deriving from the *OEUnaryBondPred* class.

Listing 7: User defined bond predicate

```
using System;
using OpenEye.OEChem;

public class UserPredBond
{
    public static void Main(string[] argv)
    {
```

(continues on next page)

(continued from previous page)

```
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "CC(=O)Nc1c[nH]cc1");
        Console.WriteLine("Number of amide bonds = " +
                          OEChem.OECount(mol, new PredAmideBond()));
    }

    private class PredHasDoubleBondO : OEUnaryAtomPred
    {
        public override bool Call(OEAtomBase atom)
        {
            foreach (OEBondBase bond in atom.GetBonds())
            {
                if (bond.GetOrder() == 2 && bond.GetNbr(atom).IsOxygen())
                {
                    return true;
                }
            }

            return false;
        }

        public override OEUnaryAtomBoolFunc CreateCopy()
        {
            OEUnaryAtomBoolFunc copy = new PredHasDoubleBondO();
            copy.ReleaseOwnership();
            return copy;
        }
    }

    private class PredAmideBond : OEUnaryBondPred
    {
        public override bool Call(OEBondBase bond)
        {
            if (bond.GetOrder() != 1)
            {
                return false;
            }

            OEAtomBase atomB = bond.GetBgn();
            OEAtomBase atomE = bond.GetEnd();
            OEUnaryAtomBoolFunc pred = new PredHasDoubleBondO();

            if (atomB.IsCarbon() && atomE.IsNitrogen() && pred.Call(atomB))
            {
                return true;
            }

            if (atomB.IsNitrogen() && atomE.IsCarbon() && pred.Call(atomE))
            {
                return true;
            }
```

(continues on next page)

(continued from previous page)

```
            return false;
        }

        public override OEUnaryBondBoolFunc CreateCopy()
        {
            OEUnaryBondBoolFunc copy = new PredAmideBond();
            copy.ReleaseOwnership();
            return copy;
        }
    }
}
```

The output of the preceding program is the following:

```
Number of amide bonds = 1
```

One advantage of functors over function pointers is that they can hold state. Since this state is held by the instance of the object it can be thread safe (unlike static-variables inside functions used with function pointers). The state of a functor can be initialized at construction. For instance, *OEHasAtomicNum* functor takes an argument on construction which defines which atomic number is required for the functor to return true.

Listing 8: User defined atom predicate with state

```
using System;
using OpenEye.OEChem;

public class UserPredAtomArg
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cc[nH]c1CC2COCNC2");
        uint[] alist = new uint[2];
        alist[0] = OEElemNo.O;
        alist[1] = OEElemNo.N;
        Console.WriteLine("Number of oxygen or nitrogen atoms = " +
                          OEChem.OECount(mol, new PredAtomicNumList(alist)));
    }

    private class PredAtomicNumList : OEUnaryAtomPred
    {
        private uint[] alist;

        public PredAtomicNumList(uint[] a)
        {
            this.alist = (uint[])a.Clone();
        }

        public override bool Call(OEAtomBase atom)
        {
            foreach (uint i in this.alist)
            {
```

(continues on next page)

(continued from previous page)

```
                if (atom.GetAtomicNum() == i)
                {
                    return true;
                }
            }

            return false;
        }

        public override OEUnaryAtomBoolFunc CreateCopy()
        {
            OEUnaryAtomBoolFunc copy = new PredAtomicNumList(this.alist);
            copy.ReleaseOwnership();
            return copy;
        }
    }
}
```

Functor substructure-based matching

The *Listing 6* shows an example how to create a user-defined atom predicate. **OEChem TK** also provides a functor template, called *OEMatchFunc*, that allows convenient substructure-based atom matching.

In the following example functors are initialized with a SMARTS string. These functors return true only if the atom matches the substructure pattern specified in construction.

Listing 9: Functor substructure-based matching

```
using System;
using OpenEye.OEChem;

public class FunctorsOEMatch
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "C1(Cl)C(N)C(F)OC1C(=O)NCCCN");
        Console.WriteLine("Number of non-amide nitrogen = " +
                          OEChem.OECount(mol, new OEMatchAtom("[N;!$(NC=O)]")));
        Console.WriteLine("Number of 5-membered ring oxygen = " +
                          OEChem.OECount(mol, new OEMatchAtom("[O;r5]")));
        Console.WriteLine("Number of carbon attached to halogen = " +
                          OEChem.OECount(mol, new OEMatchAtom("[#6][Cl,Br,F]")));
        return 0;
    }
}
```

The output of *Listing 9* is the following:

```
Number of non-amide nitrogen = 2
Number of 5-membered ring oxygen = 1
Number of carbon attached to halogen = 2
```

Molecule Partitioning

The *OESubsetMol* function can take any atom predicate as an argument and generate a subset molecule from only atoms for which the specified predicate returns true. In the following example, ring atoms are extracted from a molecule by using the *OEAtomIsInRing* atom functor.

Listing 10: Ring system extraction

```
using System;
using OpenEye.OEChem;

public class FunctorsOESubsetMol
{
    public static void Main(string[] argv)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cc[nH]c1CC2COCNC2");
        OEGraphMol submol = new OEGraphMol();
        OEChem.OESubsetMol(submol, mol, new OEAtomIsInRing(), true);
        Console.WriteLine(OEChem.OEMolToSmiles(submol));
    }
}
```

The output of *Listing 10* is the following:

```
c1cc[nH]c1.C1CNCOC1
```

In the following example, ring systems are extracted from a molecule by using *OEPartPred* functor.

Listing 11: Ring system extraction

```
using System;
using OpenEye.OEChem;

public class FunctorsOEPartPred
{
    public static int Main(string[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1cc[nH]c1CC2COCNC2");

        uint[] rings = new uint[mol.GetMaxAtomIdx()];
        uint nrrings = OEChem.OEDetermineRingSystems(mol, rings);
        OEPartPredAtom pred = new OEPartPredAtom(rings, (uint)rings.Length);
        Console.WriteLine("Number of rings = " + nrrings);
        for (uint r = 1; r < nrrings + 1; ++r)
        {
            pred.SelectPart(r);
            OEGraphMol ringmol = new OEGraphMol();
            OEChem.OESubsetMol(ringmol, mol, pred, true);
            Console.WriteLine(r + " -> " + OEChem.OEMolToSmiles(ringmol));
        }
        return 0;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

The output of *Listing 11* is the following:

```
Number of rings = 2
1 -> c1cc[nH]c1
2 -> C1CNCOC1
```

2.1.25 OEMolRecord

See also

- *OERecord* chapter

OEMolRecord is a specialized *OERecord* with some extra methods for handling molecules. The presence of convenience methods for handling molecules is the only difference between an *OEMolRecord* and an *OERecord*. The underlying representation is identical, and either class can be copy-constructed from the other with no loss of information.

The following example shows how an OEMolRecord can be used, and illustrates that OEMolRecords are just OERecords with some additional methods.

Conformer data on records

OEMolRecord also has methods for managing data for conformers. Each conformer of a molecule can get its own record to store conformer-specific data onto.

Reference Appendices

2.1.26 Periodic Table Functions

To simplify the task of dealing with the elements of the periodic table, **OEChem TK** contains several functions to obtain useful properties of the elements.

Atomic Symbol/Atomic Number

A common task is to obtain or display the atomic symbol of an atom represented by an *OEAtomBase*. To save space and reduce redundancy and consistency issues, the *OEAtomBase* class contains only an `unsigned integer` representing the atom's atomic number. This value may be obtained using the *OEAtomBase.GetAtomicNum* method. This value can be converted into an atomic symbol using the *OEGetAtomicSymbol* function.

```
string symbol = OEChem.OEGetAtomicSymbol(OEElemNo.C);
Console.WriteLine("The atomic symbol for carbon is " + symbol);
```

The example above uses the integer constant *OEElemNo.C* from the *OEElemNo* namespace. This namespace represents the atomic numbers of the 111 elements as their symbols as a convenience.

The inverse of *OEGetAtomicSymbol*, *i.e.* obtaining the atomic numbers from an atomic symbol, is performed by the *OEGetAtomicNum* function.

```
Console.WriteLine("The atomic number for 'Cl' is " + OEChem.OEGetAtomicNum("Cl"));
```

Element Properties

OEChem TK provides several functions for obtaining properties of the elements, all of which take an unsigned integer argument representing the element's atomic number.

Property Name	Return Value	Function
Average Atomic Weight	double	*OEGetAverageWeight*
Most Abundant Isotope	unsigned int	*OEGetDefaultMass*

The **OEChem TK** library also provides the following two functions to deal with specific isotopes:

- *OEGetIsotopicWeight*
- *OEIsCommonIsotope*

Both functions take an unsigned integer representing the isotope's atomic number (number of protons), and an unsigned integer representing the isotope's mass (number of protons plus neutrons).

The following example shows how to calculate molecular weight (*OECalculateMolecularWeight*) :

```
public static double CalculateMolecularWeight(OEMolBase mol)
{
    uint elemno = 0;
    uint mass = 0;
    uint implicitH = 0;
    double weight = 0.0;

    foreach (OEAtomBase atom in mol.GetAtoms())
    {
        elemno = atom.GetAtomicNum();
        mass = atom.GetIsotope();
        implicitH += atom.GetImplicitHCount();
        if ((elemno != 0) && (mass != 0) &&
             OEChem.OEIsCommonIsotope(elemno, mass))
        {
            weight += OEChem.OEGetIsotopicWeight(elemno, mass);
        }
        else
        {
            weight += OEChem.OEGetAverageWeight(elemno);
        }
    }

    weight += implicitH * OEChem.OEGetAverageWeight(OEElemNo.H);
    return weight;
}
```

Atom Radius Functions

Property Name	Get Method	Applied on *OEMolBase*	Reference
Covalent Radius	*OEGetCovalentRadius*	*OEAssignCovalentRadii*	
Default Radius used by DelPhi	*OEGetDelphiRadius*	*OEAssignDelphiRadii*	Accelrys
Effective Ionic Radius	*OEGetHonigIonicCavityRa*	*OEAssignHonigIonicCavityRadi*	[Rashin-1985]
Van der Waals Radius	*OEGetBondiVdWRadius*	*OEAssignBondiVdWRadii*	[Bondi-1964]
Van der Waals Radius	*OEGetPaulingVdWRadius*	*OEAssignPaulingVdWRadii*	[Pauling-1960]
Van der Waals Radius	N/A [1]	*OEAssignZap9Radii*	[Nicholls-2008]
Van der Waals Radius		*OEAssignZap7Radii*	[Nicholls-2010]
Generic Assignment Function	N/A [2]	*OEAssignRadii*	

Table footnote:

[1] The get method is not available, since the radius value of *OEElemNo.O* and *OEElemNo.N* (defined in [Nicholls-2008]) depend on the chemical environment of the given atom.

[2] Can be used to call any of the other methods to assign radii as well as to clear atomic radii.

The following snippet shows how to set and access the "Pauling" van der Waals radius in a given *OEMolBase*:

```
OEChem.OEAssignPaulingVdWRadii(mol);
foreach (OEAtomBase atom in mol.GetAtoms())
{
    Console.WriteLine(atom.GetIdx() + " " + atom.GetRadius());
}
```

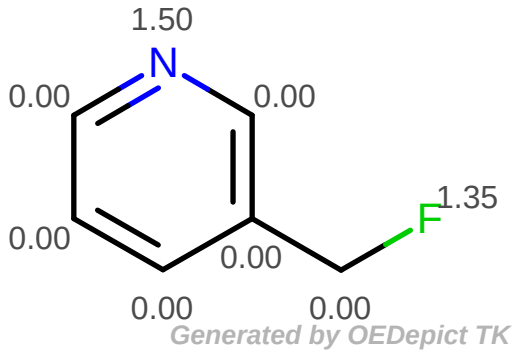


Fig. 67: *Example of setting "Pauling" van der Waals radius*

2.1.27 SMILES Line Notation

A SMILES syntax overview can be found at the documentation of SMILES on the Daylight Chemical Information Systems site

Unfortunately, there are a number of ambiguities in the original paper describing the Daylight SMILES syntax (refs

below). These have led to different SMILES being accepted or rejected by different SMILES parser implementations. The following historical table illustrates some of these corner-cases. These types of cases continue to be an issue with SMILES parsers.

Table 14: **Differences between various SMILES parser implementations**

SMILES	Daylight 4.4-4.9	Corina 1.6	Corina WWW	Concord 3.2.1	COBRA 3.21A	Synopsis 4.0	OEChem 5.0.0.0
`C1.C1`	Y	Y	Y	N	N	Y	Y
`C%00CC%0`	Y	Y	Y	N	N	N	Y
`C(C.C)C`	Y	Y	Y	N	N	Y	Y
`C(C)1CC1`	Y	N	N	N	Y	N	Y
`C(.C)`	Y	Y	Y	N	N	Y	Y
`C()`	Y	Y	N	Y	Y	Y	Y
`(CO)=O`	N	N	N	N	N	Y	Y
`(C)`	N	N	N	N	N	Y	Y
`.C`	N	N	N	Y	Y	N	Y
`C..C`	N	Y	N	Y	Y	N	Y
`C.`	N	Y	Y	Y	Y	Y	Y
`C=(O)C`	N	Y	N	N	Y	N	N
`C(((C))`	N	Y	N	Y	N	Y	Y
`C.(C)`	N	Y	N	Y	N	N	Y
`C1CC(=1)`	N	Y	N	N	N	N	Y
`C1CC(1)`	N	N	N	N	N	N	Y
`C(C.)`	N	Y	N	N	N	N	Y
`C==C`	N	Y	N	N	N	N	Y
`C(1CC1)`	N	N	N	N	N	N	Y
`C(1)CC1`	N	N	N	N	N	N	Y

Because of these types of SMILES ambiguities, the **OEChem TK** SMILES parser has two modes. The default is *`relaxed`*, which is more forgiving and accepts SMILES as per the table above and enables the SMILES extensions described in the next section. The SMILES parser also has a *`strict`* mode that is far less forgiving about dubious SMILES strings.

Extensions to Daylight SMILES

The **OEChem TK** SMILES parsers support several minor extensions to Daylight syntax. Each of these extensions and its motivations are listed below.

Quadruple Bond

In addition to `-`, `=` and `#` for specifying single, double and triple bonds respectively, **OEChem TK** also supports `$` for specifying quadruple bonds. An example would be octachlorodirhenate (III), which is written as `[Rh-](Cl)(Cl)(Cl)(Cl)$[Rh-](Cl)(Cl)(Cl)Cl`.

Unquoted and Additional Elements

In addition to the standard Daylight unquoted elements, B, C, N, O, F, P, S, Cl, Br and I, **OEChem TK**'s SMILES readers also allow `H`, `D` and `T` to specify hydrogen, deuterium and tritium. Additionally, to support Syracuse SMILES, 'CL' and 'BR' are considered 'Cl' and 'Br'. The periodic table is also extended from 102 to 109 elements, *i.e.* `[Sg]` for Seaborgium, with the addition of `[D]` and `[T]` representing `[2H]` and `[3H]` respectively.

OEChem TK may support `Na`, `Li`, and `K` as unquoted elements to support Syracuse SMILES at some point in the future.

Aromatic Tellurium

In order to support OpenEye's aromaticity model, which allows Tellurium to be aromatic, the SMILES parser has been extended to support `[te]`, such as in tellurophene, `[te]1cccc1`, which follows in the sequence furan (`o1cccc1`), thiophene (`s1cccc1`) and selenophene (`[se]1cccc1`).

Atom Maps in Molecules

Traditionally, SMILES atom maps, *i.e.* `[Pb:1]`, are only ever used and specified in reaction molecules, `[Pb:1]>>[Au:1]`. However, **OEChem TK** extends this notion to allow atom maps to be used in discrete molecules. This is often useful for denoting significant sites or attachment points in a molecule. Traditionally in SMILES, isotopes of element zero have been used to perform this role, however in **OEChem TK** both `[*:1]` and `[1*]` may be used.

RGroup Attachment Points

As a short hand to support specifying templates for combinatorial libraries, and to support existing Cactus and JChem/Marvin usage, **OEChem TK** allows `[R2]` to be used as short-hand for `[*:2]`. For inquiring minds, the SMILES `[R2:3]` is interpreted as `[*:3]` or `[R3]`, with the last specification taking priority.

External Bond Attachment Points

OEChem TK SMILES also allows supports external closures, or potentially unsatisfied *ring closures*. These have the syntax, ampersand followed by a ring closure specification, *i.e.* an optional bond order followed by either a digit or a `%` character and two digits. The index space of external bonds and ring closures is separate, so that the ring closure `2` and the external attachment point `&2` are unrelated.

When external attachment points are paired within a SMILES string, they behave identically to ring closures, using a separate index space. Hence, the SMILES `c&1ccccc&1` is interpreted the same way as `c1ccccc1`, and `C&1.C&1` is interpreted like `C1.C1`, *i.e.* the SMILES `CC`.

However, unlike ring closures, unpaired external attachment points are allowed and are interpreted like RGroup attachment points above. Hence, the SMILES `CC&1` (on its own) is equivalent to the RGroup attachment SMILES `CC[R1]`, which is equivalent to the atom mapped molecule `CC[*:1]`.

The major advantage of these semantics, inspired by Daylight's CHUCKLES, is that it allows convenient enumeration of combinatorial libraries using string concatenation. For example, three components of a library may be specified as `C&1CCC&2`, `F&1` and `Br&2`. The combined string `C&1CCC&2.F&1.Br&2` is interpreted as the reaction product, *i.e.* `FCCCCBr`.

As with ring closures, bond orders may be specified after the ampersand and before the closure index, `C&=1`, and two digit closures are indicated by a `%` prefix, *i.e.* `C&%12` or `C&=%12`.

See also

- [Weininger-1988]
- [Weininger-1989]

2.1.28 SMARTS Pattern Matching

SMARTS Syntax

SMARTS is a line notation developed by Daylight Chemical Information Systems for compactly representing molecular substructure queries. The SMARTS language can be considered an extension or generalization of Daylight's SMILES notation for representing discrete molecules.

A SMARTS syntax overview can be found at the documentation of SMARTS on the Daylight Chemical Information Systems site

Atom Primitives

Symbol	Description	Argument?	Default Value
A	non-aromatic (aliphatic) atom	no	(no default)
a	aromatic atom	no	(no default)
D<n>	degree (explicit connections)	yes	(no default)
H<n>	total hydrogen count	optional	exactly one
h<n>	implicit hydrogen count	optional	exactly one
R<n>	ring bond count [1]	optional	any ring atom
x<n>	ring bond count [2]	optional	any ring atom
r<n>	smallest ring size	optional	any ring atom
v<n>	valence (total bond order)	yes	(no default)
X<n>	connectivity (total connections)	yes	(no default)
#<n>	atomic number [3]	yes	(no default)
+<n>	positive charge	optional	+1 cation (++ is +2, etc)
-<n>	negative charge	optional	-1 anion (– is -2, etc)
^<n>	atomic hybridization [4]	yes	(no default)
@	anticlockwise local chirality	no	
@@	clockwise local chirality	no	
@<n>	chirality class	optional	anticlockwise
n explicit	atomic mass no		

Table footnote:

[1] The semantics of the ring count primitive, R, differs slightly between Daylight SMARTS and OpenEye SMARTS. In Daylight semantics, R<n> means that an atom is in n rings of the chosen SSSR. As the choice of SSSR is non-deterministic, this interpretation can cause an arbitrary set of atoms to match depending upon input order. For example, in the symmetric molecule, cubane, four of the eight atoms will appear in two SSSR rings, and half of the atoms appear in three, but the choice is made almost randomly. Rather than attempt to reproduce these weak semantics, OpenEye strengthens the definition of R<n> to mean the number of ring bonds to an atom, which is graph invariant and therefore independent of a molecule's input order. Notice, that the interpretation of [R] and [R0], *i.e.* ring membership, remains the same. Similarly, Daylight [R1] is approximately equal to OpenEye [R2], and Daylight [R2] is approximately equivalent to OpenEye [R3].

[2] Note that [x] was implemented by Daylight v4.9 and **OEChem TK** 1.5, and is exactly synonymous to **OEChem TK**'s [R].

[3] In OEChem, #0 indicates wildcard or attachment atoms. For example, the *OEAtomBase.GetAtomicNum* method returns 0 for R-Group atoms (for example it would match the R-groups of a molecule initialized as CC[R1], CC[R], or CC[*]). *OEAtomBase.GetAtomicNum* also returns 0 for color atoms.

[4] The atomic hybridization primitive, ^, is an **OpenEye*** extension that is not available in Daylight SMARTS, but can be implemented using recursive SMARTS.

Bond Primitives

Syntax	Description
default	single or aromatic
-	single bond (not aromatic)
=	double bond (not aromatic)
#	triple bond
~	any bond (wildcard)
:	aromatic bond
@	ring bond
/	directional single 'up' bond
\	directional single 'down' bond
/?	directional 'up' or unspecified
\?	directional 'down' or unspecified

Logical Operators

Syntax	Description
`!e`	not e
`e1 & e2`	e1 and e2 (high precedence)
`e1,e2`	e1 or e2
`e1;e2`	e1 and e2 (low precedence)

Examples

Table 15: **Example of matching any defined stereo centers**

SMARTS: `[C;$([*@]([*])([*]))]`

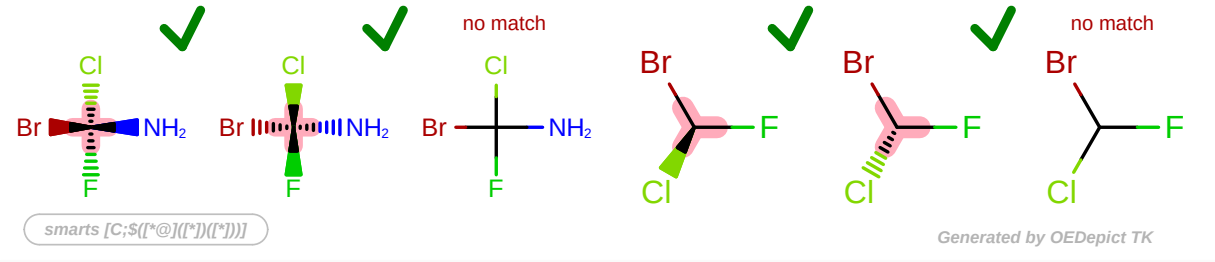
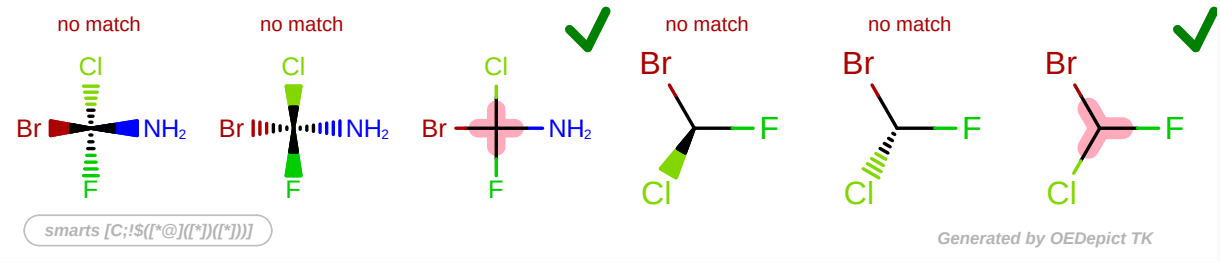


Table 16: **Example of matching any undefined stereo centers**

SMARTS: `[C;!$([*@]([*])([*]))]`



2.1.29 Atom Stereo From Bond Stereo

Tertiary

The following images depict how various tertiary bond stereo configurations are interpreted by the *OEMDLStereoFromBondStereo* function.

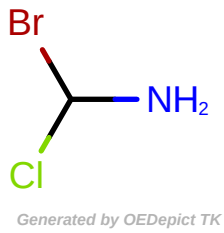


Fig. 68: Example of **unspecified** tertiary bond stereo configuration

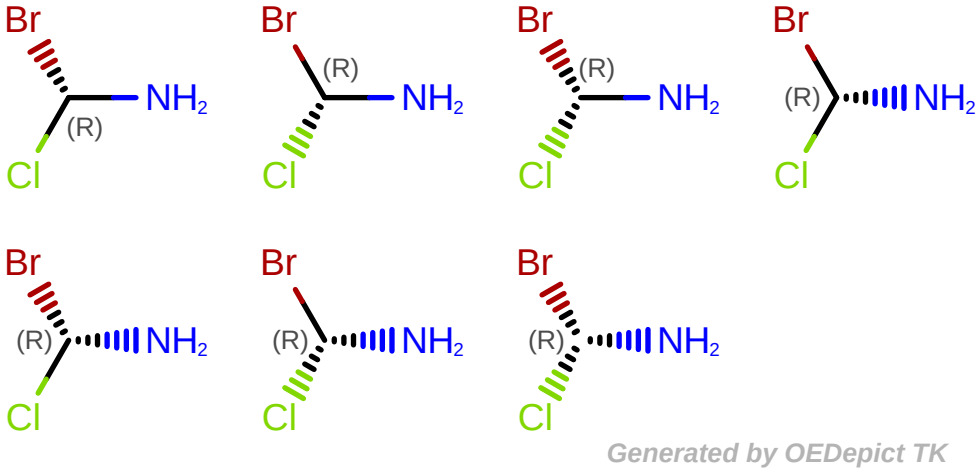


Fig. 69: Examples of **left** tertiary bond stereo configuration

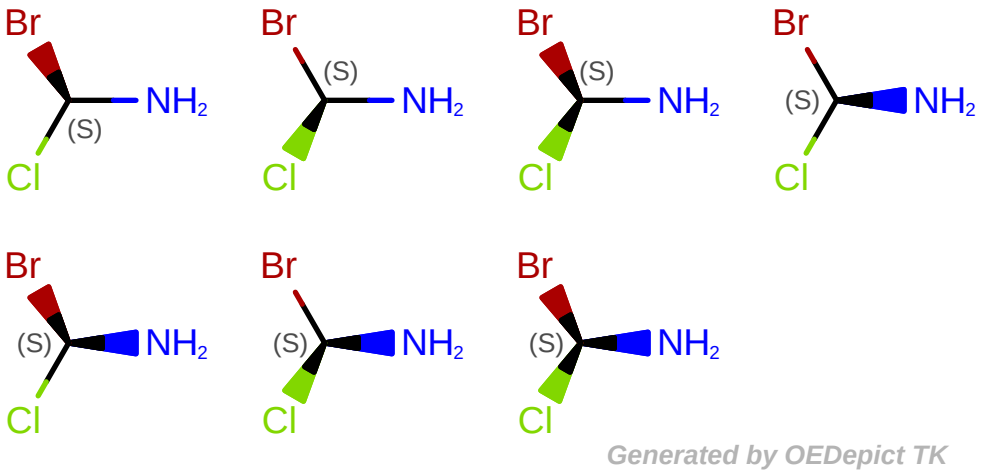


Fig. 70: Examples of **right** tertiary bond stereo configuration

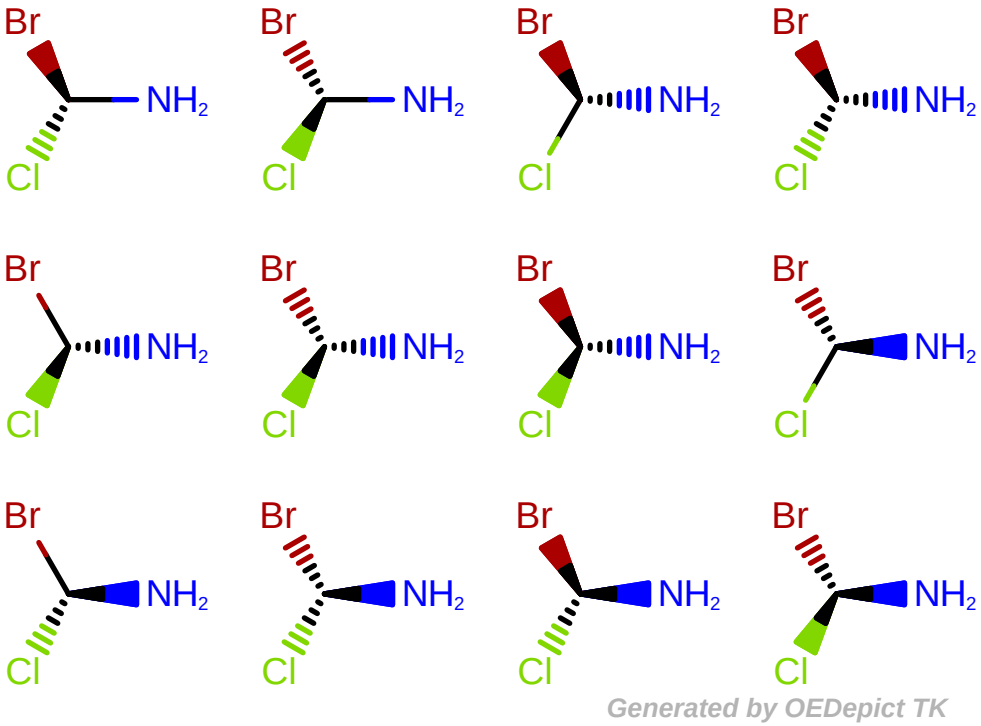


Fig. 71: Examples of **invalid** tertiary bond stereo configuration

Note

A stereo atom center with invalid bond stereo configuration is treated as *OEAtomStereo.Undefined*.

Quaternary

The following images depict how various quaternary bond stereo configurations are interpreted by the *OEMDLStereoFromBondStereo* function.

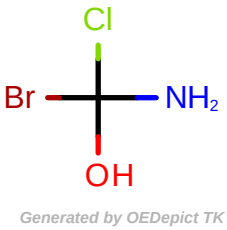


Fig. 72: Example of **unspecified** quaternary bond stereo configuration

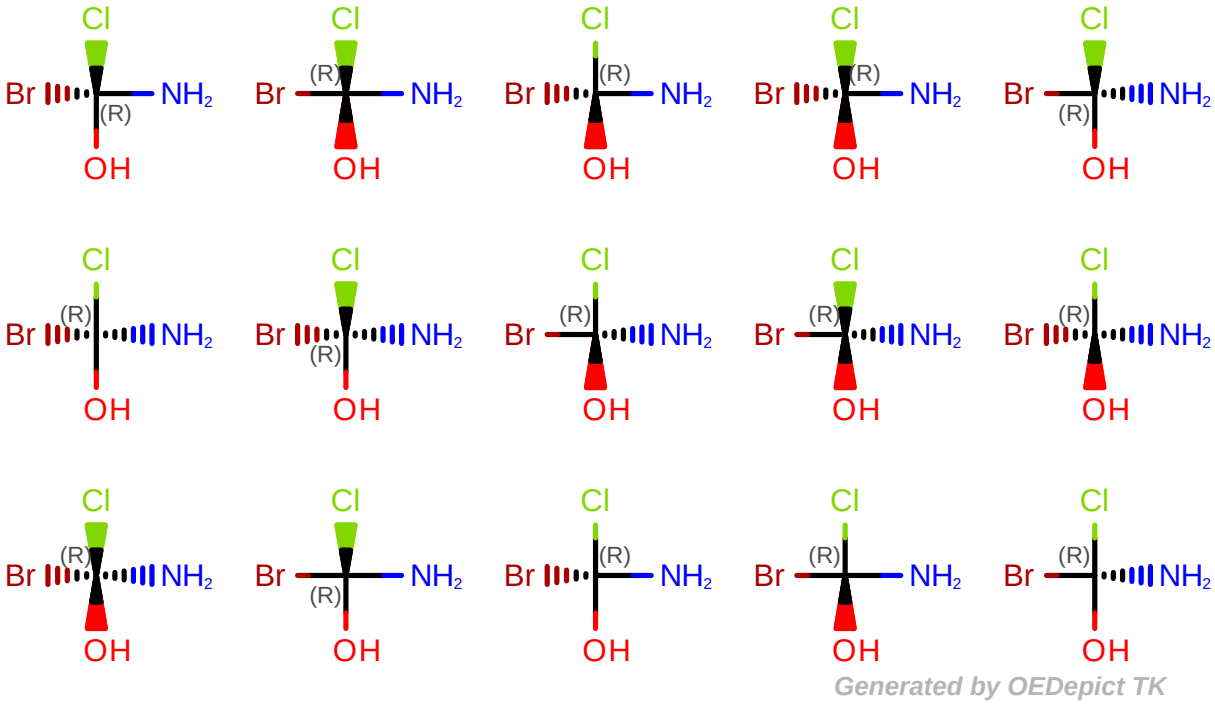


Fig. 73: Examples of **left** quaternary bond stereo configuration

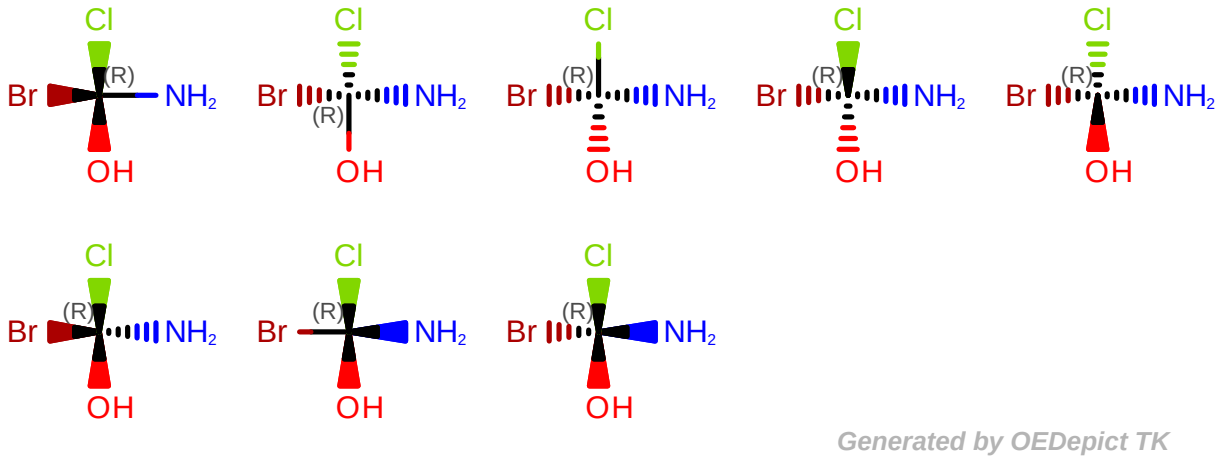


Fig. 74: Examples of **corrected left** quaternary bond stereo configurations

Note

The `OEMDLStereoFromBondStereo` function throws a warning message when detects bond stereo configurations depicted in the *Figure Examples of corrected left quaternary bond stereo configurations* and treated these atom centers as `OEAtomStereo.Left`.

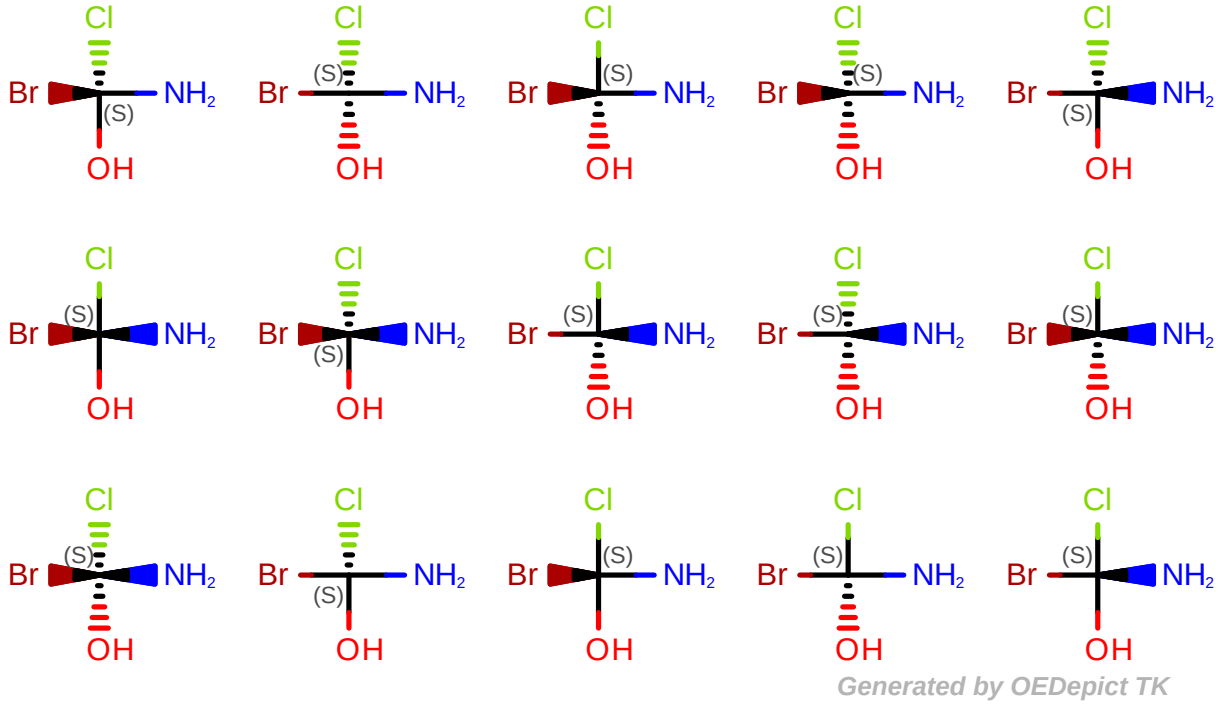


Fig. 75: Examples of **right** quaternary bond stereo configuration

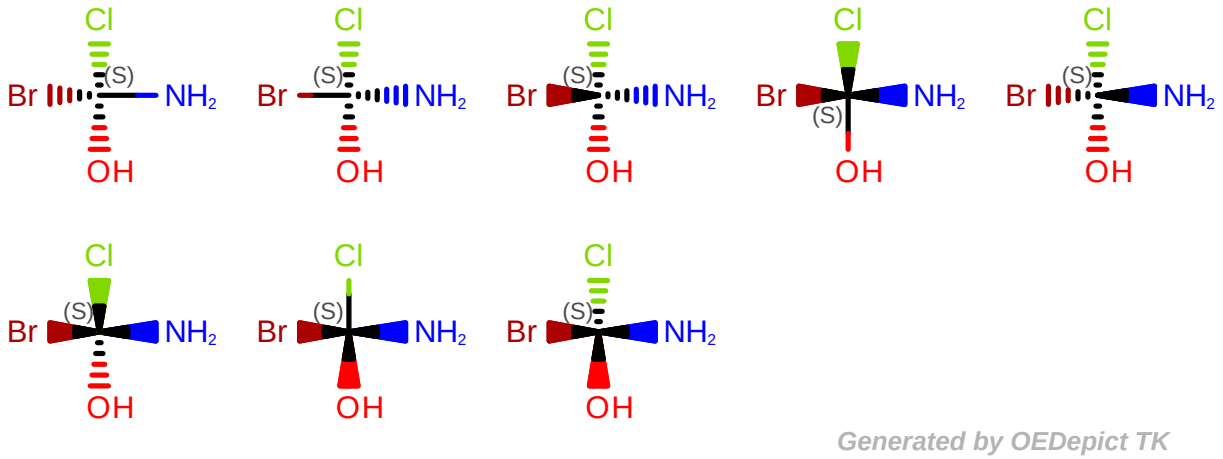


Fig. 76: Examples of **corrected right** quaternary bond stereo configurations

Note

The `OEMDLStereoFromBondStereo` function throws a warning message when detects bond stereo configurations depicted in the *Figure Examples of corrected right quaternary bond stereo configurations* and treated these atom centers as `OEAtomStereo.Right`.

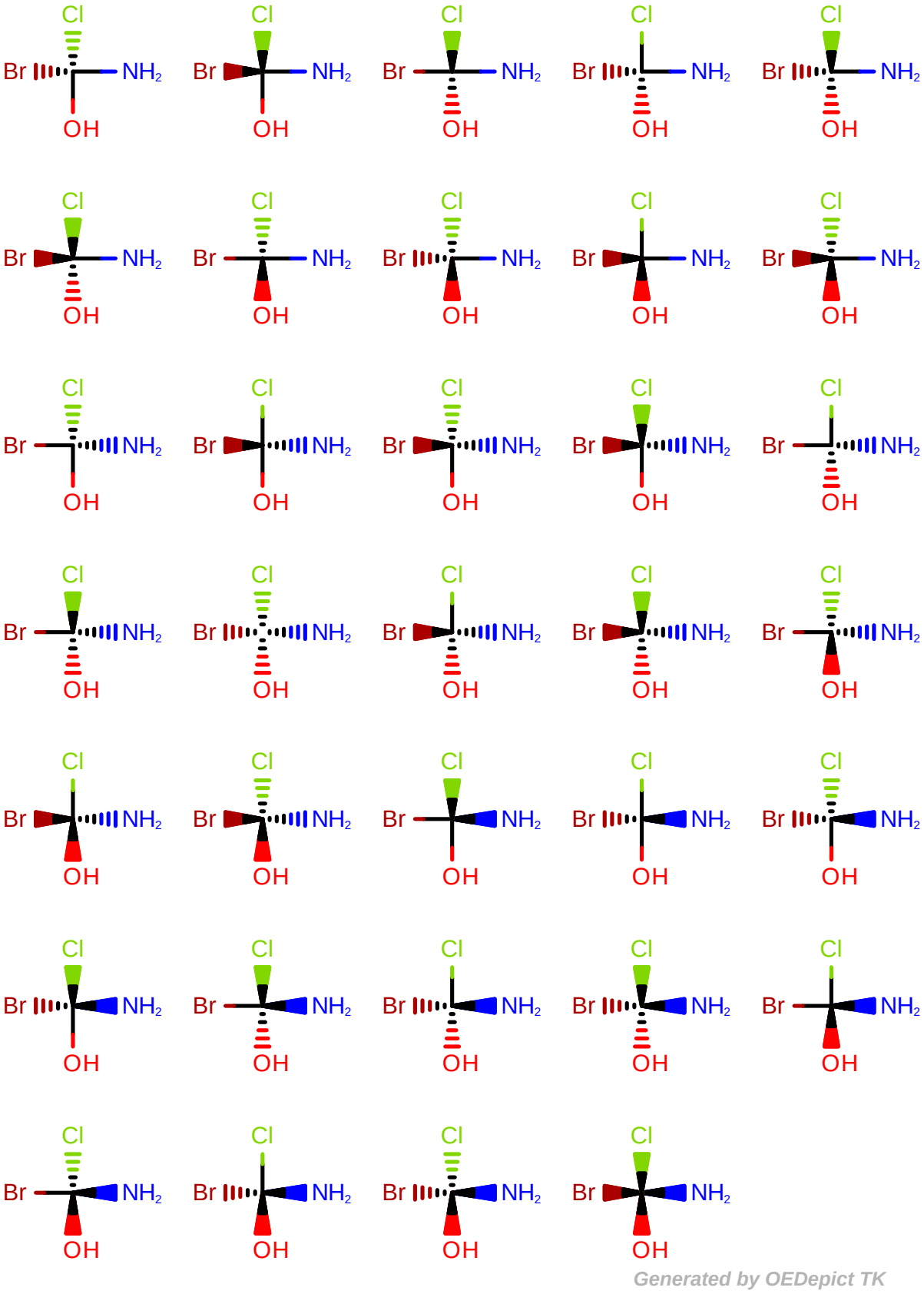


Fig. 77: Examples of **invalid** quaternary bond stereo configuration

Note

A stereo atom center with invalid bond stereo configuration is treated as *OEAtomStereo.Undefined*.

See also

- stereo atom
- *OEAtomBase.GetStereo* method
- *OEAtomBase.SetStereo* method
- *OEAtomBase.HasStereoSpecified* method

2.1.30 InChI validation

The InChI strings returned by *OECreateInChI* were rigorously tested to ensure that they were identical to the InChI strings generated by the standalone InChI utility program.

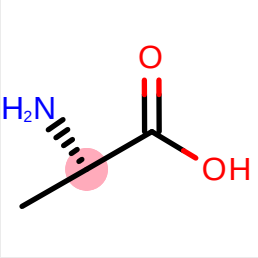
Table 17: Performance of **OECreateInChI** to generate *standard InChI*

Database	Size	Number of failures	Success Rate
eMolecules	9.1M	1,350	99.97 %
ChEMBL_23	1.7M	8,255	99.52 %

Many failures are due to the problem that **OEChem TK** and *InChI* handle some incorrect bond stereo configurations differently. See example in *Difference due atom stereo perception*

Additionally there are approximately 7,300 structures in ChEMBL_23 where the InChI, `SMILES` and SDF forms for the record are in disagreement resulting in ambiguity for the expected or reference InChI.

Table 18: **Differences due atom stereo perception**

	
InChI standalone	InChI=1S/C3H7NO2/c1-2(4)3(5)6/h2H,4H2,1H3,(H,5,6)
OECreateInCl	InChI=1S/C3H7NO2/c1-2(4)3(5)6/h2H,4H2,1H3,(H,5,6)/t2-/m1/s1

See example in *Difference due atom stereo perception*

2.2 OESystem Theory

2.2.1 Command Line Interface and Options

This chapter describes how **Options** classes that are derived from *OEOptions* can be configured with command line inputs passed in through `argv`.

The *OEConfigureOpts* function is a convenience function that takes an *OEOptions* derivative options class and the command line arguments `argv`, parses the command line arguments, and configures the option parameters with command line inputs. A *OEInterface* is generated internally from the *OEOptions*, which performs the command lines parsing.

Note that this only applies to **Options** classes that are derived from the *OEOptions*. There are a handful of **Options** classes that still does not derive from *OEOptions*, and hence cannot be configured with command line inputs directly. *Command Line Interface Definitions* is an alternative way to work with command line interfaces for such cases.

In addition to the convenience function *OEConfigureOpts*, two additional general purpose **Options** classes, *OESimpleAppOptions*, and *OERefInputAppOptions*, are provided that extends any other options classes to include necessary input/output parameters that are generally required to create an application.

See also

- *OEConfigureOpts* function
- *OEOptions* class
- *OESimpleAppOptions* class
- *OERefInputAppOptions* class

Application with Input, output, and Options

The following example shows how to use the *OESimpleAppOptions* class to extend an **Options** class with an additional input and output file.

The specific example is taken from **OEDocking**, and uses the the `OEMakeReceptor` function to make a receptor in a design unit. In this example, the OEMakeReceptorOptions is extended with an input and output.

Listing 1: Application with input, output, and options (Making Receptor)

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
```

(continues on next page)

(continued from previous page)

```
*/
using System;
using OpenEye.OEChem;
using OpenEye.OEBio;
using OpenEye.OEDocking;

public class MakeReceptor
{
  public static void Main(string[] argv)
  {
    OEMakeReceptorOptions recOpts = new OEMakeReceptorOptions();
    OESimpleAppOptions opts = new OESimpleAppOptions(recOpts, "MakeReceptor",␣
↪OEFileStringType.DU, OEFileStringType.DU);

    if (OEChem.OEConfigureOpts(opts, argv, false) == OEOptsConfigureStatus.Help)
      Environment.Exit(1);

    recOpts.UpdateValues(opts);

    oeifstream ifs = new oeifstream(opts.GetInFile());
    oeofstream ofs = new oeofstream(opts.GetOutFile());

    OEDesignUnit du = new OEDesignUnit();
    OEDesignUnit receptor = new OEDesignUnit();
    while (OEBio.OEReadDesignUnit(ifs, du))
    {
      if (OEDocking.OEMakeReceptor(du, recOpts))
        OEBio.OEWriteDesignUnit(ofs, du);
      else
        OEChem.OEThrow.Fatal(du.GetTitle() + " failed to make a receptor");

    }
  }
}
```

Application with Reference system, Input, output, and Options

Besides an input and output, some applications require an additional reference system input. The *OERefInputAppOptions* class can be used to extend an options class with three additional parameters, an input, output, and a reference system.

The following code example is taken from **OEDocking**, and uses the OEDock object to perform docking. In this example, the OEDockOptions is extended with an input, output, and a **receptor** (i.e., the reference system).

Listing 2: Application with input, reference, output, and options (Docking Molecules)

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
```

(continues on next page)

(continued from previous page)

```
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
using System;
using OpenEye.OEChem;
using OpenEye.OEBio;
using OpenEye.OEDocking;

public class DockMolecules
{
  public static void Main(string[] argv)
  {
    OEDockOptions dockOpts = new OEDockOptions();
    OERefInputAppOptions opts = new OERefInputAppOptions(dockOpts, "DockMolecules",␣
↪OEFileStringType.Mol3D,
            OEFileStringType.Mol3D, OEFileStringType.DU, "-receptor");

    oemolistream ifs = new oemolistream();
    if (!ifs.open(opts.GetInFile()))
      OEChem.OEThrow.Fatal("Unable to open " + opts.GetInFile() + " for reading");
    oemolostream ofs = new oemolostream();
    if (!ofs.open(opts.GetOutFile()))
      OEChem.OEThrow.Fatal("Unable to open " + opts.GetOutFile() + " for writing");
    oeifstream rfs = new oeifstream();
    if (!rfs.open(opts.GetRefFile()))
      OEChem.OEThrow.Fatal("Unable to open " + opts.GetRefFile() + " for reading");

    OEDesignUnit du = new OEDesignUnit();
    if (!OEBio.OEReadDesignUnit(opts.GetRefFile(), du))
      OEChem.OEThrow.Fatal("Failed to read design unit ");
    if (!du.HasReceptor())
      OEChem.OEThrow.Fatal("Design unit " + du.GetTitle() + " does not contain a receptor
↪");

    OEDock dock = new OEDock(dockOpts);
    dock.Initialize(du);

    foreach (OEMol mcmol in ifs.GetOEMols())
    {
      Console.WriteLine("docking {0}\n", mcmol.GetTitle());
      OEGraphMol dockedMol = new OEGraphMol();
      uint retCode = dock.DockMultiConformerMolecule(dockedMol, mcmol);
      if (retCode != OEDockingReturnCode.Success)
```

(continues on next page)

(continued from previous page)

```
      {
        OEChem.OEThrow.Error("Docking failed with error code " + OEDocking.
↪OEDockingReturnCodeGetName(retCode));
      }
      String sdtag = OEDocking.OEDockMethodGetName(dockOpts.GetScoreMethod());
      OEDocking.OESetSDScore(dockedMol, dock, sdtag);
      dock.AnnotatePose(dockedMol);
      OEChem.OEWriteMolecule(ofs, dockedMol);
    }
    ofs.close();
    ifs.close();
  }

}
```

Application with Input and output only

Some applications do not have any additional options or parameters, and only require an input and output. The *OESimpleAppOptions* class has an overload that does not require an *OEOptions* input argument.

The following code example is taken from **OEQuacpac**, and uses the `OEAssignCharges` method to assign charges in an *OEDesignUnit*. This example only configures an input and output, and does not have any additional parameters to configure.

Adding custom parameters

Custom parameters can also be added to an application API when a built-in **Options** class does not contain all required parameters for the application.

The following code example is taken from **OEDocking**, and uses the OEPosit object to generate pose structures. In this example, the OEPositOptions is extended with an input, output, and **receptor** (i.e., the reference system). However, the number of poses to generate is not a built-in parameter into the OEPositOptions. This example shows how to add an additional parameter, for the number of poses, to be configured through the command line.

Listing 4: Adding custom parameters (POSIT with Multiple Receptors)

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
```

(continues on next page)

(continued from previous page)

```
*/
using System;
using OpenEye.OEChem;
using OpenEye.OEBio;
using OpenEye.OEDocking;

public class MyOptions : OEPositOptions
{
    public MyOptions() : base()
    {
        OEUIntParameter param1 = new OEUIntParameter("-numPoses", 1);
        param1.AddLegalRange("1", "20");
        param1.SetBrief("Number of poses to generate");
        _param1 = AddParameter(param1);
    }

    public uint GetNumPoses()
    {
        if (_param1.GetHasValue())
            return (uint.Parse(_param1.GetStringValue()));
        return (uint.Parse(_param1.GetStringDefault()));
    }

    OEParameter _param1;
}

public class PoseMolecules
{
    public static void Main(string[] argv)
    {
        MyOptions positOpts = new MyOptions();
        OERefInputAppOptions opts = new OERefInputAppOptions(positOpts, "PoseMolecules",␣
↪OEFileStringType.Mol3D,
            OEFileStringType.DU, OEFileStringType.DU, "-receptor");

        if (OEChem.OEConfigureOpts(opts, argv, false) == OEOptsConfigureStatus.Help)
            Environment.Exit(1);

        positOpts.UpdateValues(opts);

        oemolistream ifs = new oemolistream();
        if (!ifs.open(opts.GetInFile()))
            OEChem.OEThrow.Fatal("Unable to open " + opts.GetInFile() + " for reading");

        oeifstream rfs = new oeifstream();
        if (!rfs.open(opts.GetRefFile()))
            OEChem.OEThrow.Fatal("Unable to open " + opts.GetRefFile() + " for reading");

        oeofstream ofs = new oeofstream();
        if (!ofs.open(opts.GetOutFile()))
            OEChem.OEThrow.Fatal("Unable to open " + opts.GetOutFile() + " for writing");
```

(continues on next page)

(continued from previous page)

```
        OEPosit poser = new OEPosit(positOpts);

        OEDesignUnit du = new OEDesignUnit();
        uint count = 0;
        while (OEBio.OEReadDesignUnit(rfs, du))
        {
            if (!du.HasReceptor())
                OEChem.OEThrow.Fatal("Design unit " + du.GetTitle() + " does not contain␣
↪a receptor");
            poser.AddReceptor(du);
            count += 1;
        }
        if (count == 0)
            OEChem.OEThrow.Fatal("Input design unit does not contain any receptor");

        OEChem.OEThrow.Info("Number of conformers: " + positOpts.GetNumPoses());
        OEChem.OEThrow.Info("Best receptor pose flag: " + positOpts.
↪GetBestReceptorPoseOnly());

        foreach (OEMol mcmol in ifs.GetOEMols())
        {
            OEChem.OEThrow.Info("posing " + mcmol.GetTitle());
            OEPositResults results = new OEPositResults();
            uint returnCode = poser.Dock(results, mcmol, positOpts.GetNumPoses());

            if (returnCode == OEDockingReturnCode.Success)
            {
                foreach (OESinglePoseResult result in results.GetSinglePoseResults())
                {
                    OEDesignUnit posedDU = new OEDesignUnit(result.GetDesignUnit());
                    posedDU.SetDoubleData(poser.GetName(), result.GetProbability());
                    OEChem.OEThrow.Info("Receptor used: " + posedDU.GetTitle() + " pose␣
↪probability: " + result.GetProbability());
                    OEBio.OEWriteDesignUnit(ofs, posedDU);
                }
            }
            else
            {
                string errMsg = OEDocking.OEDockingReturnCodeGetName(returnCode);
                OEChem.OEThrow.Warning(mcmol.GetTitle() + " : " + errMsg);
            }
        }

        ofs.close();
        ifs.close();
        rfs.close();
    }
}
```

Defining new options class

Options classes, derived from *OEOptions*, can also be defined at the script level, and wrapped with the *OESimpleAppOptions* or *OERefInputAppOptions*, to build applications.

The following code example is taken from **OEFF**, and optimizes molecule conformations using a force field. In this example, a new **Options** class is defined to accept command line input for the force field.

2.2.2 Command Line Interface Definition

When not working with **Options** derived from *OEOptions*, *OEInterface* class can still be used to create a command line application. For such usage, *OEInterface* has a convenience constructor that takes an interface definition and a command line to parse. The constructor will configure the interface with the specified definition, then parse the options from command line passed in through `argv`.

Listing 1: Getting started with OEInterface command line parsing

```
using System;
using OpenEye.OEChem;

public class GettingStartedWithOEInterface
{
    static string InterfaceData = @"
!PARAMETER -b
  !TYPE bool
  !BRIEF A boolean parameter
!END
!PARAMETER -i
  !TYPE int
  !BRIEF An integer parameter
!END
!PARAMETER -f
  !TYPE float
  !BRIEF A float parameter
!END
!PARAMETER -str
  !TYPE string
  !BRIEF A string parameter
!END
";

    public static int Main(string[] args)
    {
        OEInterface itf = new OEInterface(InterfaceData, "GettingStartedWithOEInterface",
↪ args);

        Console.WriteLine("-b = " + itf.GetBool("-b"));
        Console.WriteLine("-i = " + itf.GetInt("-i"));
        Console.WriteLine("-f = " + itf.GetFloat("-f"));
        Console.WriteLine("-str = " + itf.GetString("-str"));
        return 0;
    }
}
```

The values specified on the command line are then accessed on the *OEInterface* object using a `Get` method that corre-

sponds to the type of the parameter. The following is an example of executing the program in *Listing 1*:

Warning

The convenience constructor will call the system exit function forcing the program to exit in the following situations:

- The interface definition is malformed
- The user has requested help on any of the parameters, *e.g.*, `--help -str`
- The command line is malformed, *i.e.*, missing or extra parameters

For these reasons the convenience constructor may not be suitable for use in a library setting, it is intended for standalone applications.

Parameter Value Access

The `Get` methods of the *OEInterface* returns the value of the specified parameter the user entered on the command line, or in the case that the parameter was not specified by the user on the command line the default value specified in the interface definition is returned. If there is no default or user specified value the default constructed object is returned (*i.e.*, 0.0f for `float` parameters and an empty string for `string` parameters) and a warning is issued. For example, the following demonstrates how running the program in *Listing 1* without any arguments will display a lot of warnings about uninitialized values being used.

The warnings can be avoided by using the *OEInterface.Has* method to check if there is no default value or user specified value. *Listing 2* demonstrates how to check for the existence of certain parameters.

Listing 2: Testing for parameter existence with the Has method

```
using System;
using OpenEye.OEChem;

public class UsingOEInterfaceHas
{
    static string InterfaceData = @"
!PARAMETER -b
  !TYPE bool
  !BRIEF A boolean parameter
!END
!PARAMETER -i
  !TYPE int
  !BRIEF An integer parameter
!END
!PARAMETER -f
  !TYPE float
  !BRIEF A float parameter
!END
!PARAMETER -str
  !TYPE string
  !BRIEF A string parameter
!END
";

    public static void Main(string [] argv)
    {
```

(continues on next page)

(continued from previous page)

```
        OEInterface itf = new OEInterface(InterfaceData, "UsingOEInterfaceHas", argv);

        if(itf.HasBool("-b"))
            Console.WriteLine("-b = " + itf.GetBool("-b"));
        if(itf.HasInt("-i"))
            Console.WriteLine("-i = " + itf.GetInt("-i"));
        if(itf.HasFloat("-f"))
            Console.WriteLine("-f = " + itf.GetFloat("-f"));
        if(itf.HasString("-str"))
            Console.WriteLine("-str = " + itf.GetString("-str"));
    }
}
```

The following demonstrates the output of the program in *Listing 2*. Notice that only variables present on the command line will trigger *OEInterface.Has* to return `true`. This is an important distinction when dealing with `boolean` parameters, *OEInterface.Has* will not return the value of the `boolean` parameter, just the existence of it.

Default Values

Using *OEInterface.Has* all over a program can become quite clumsy. Therefore it is recommended to use `!DEFAULT` in the parameter interface definition to specify the parameter's default value whenever a default value can be specified. When a parameter has a default value *OEInterface.Has* will always return `true`. *Listing 3* demonstrates how to set a default value for a parameter.

Listing 3: Setting a default value for a parameter

```
using System;
using OpenEye.OEChem;


public class UsingOEInterfaceDefaults
{
    static String InterfaceData = @"
!PARAMETER -b
  !TYPE bool
  !DEFAULT false
  !BRIEF An boolean parameter
!END
!PARAMETER -i
  !TYPE int
  !DEFAULT 2
  !BRIEF An integer parameter
!END
!PARAMETER -f
  !TYPE float
  !DEFAULT 0.5
  !BRIEF A float parameter
!END
!PARAMETER -str
  !TYPE string
  !DEFAULT foo
  !BRIEF A string parameter
```

(continues on next page)

(continued from previous page)

```
!END
";

    public static void Main(string [] argv)
    {
        OEInterface itf = new OEInterface(InterfaceData, "UsingOEInterfaceDefaults",␣
↪argv);

        Console.WriteLine("-b = " + itf.GetBool("-b"));
        Console.WriteLine("-i = " + itf.GetInt("-i"));
        Console.WriteLine("-f = " + itf.GetFloat("-f"));
        Console.WriteLine("-str = " + itf.GetString("-str"));
    }
}
```

This allows the code to take a very simple structure that doesn't have to care whether the parameter was actually specified on the command line. The following is what is printed by the program in *Listing 3* when no arguments are given.

Requiring Parameters

Programs usually have a bare minimum set of parameters that must be specified. A parameter can be marked as required in the interface definition by using `!REQUIRED true` in the parameter definition. *Listing 4* demonstrates making all the parameters used in the previous example required.

Listing 4: Requiring all parameters to be specified.

```
using System;
using OpenEye.OEChem;

public class UsingOEInterfaceRequired
{
    static string InterfaceData = @"
!PARAMETER -b
  !TYPE bool
  !REQUIRED true
  !BRIEF An boolean parameter
!END
!PARAMETER -i
  !TYPE int
  !REQUIRED true
  !BRIEF An integer parameter
!END
!PARAMETER -f
  !TYPE float
  !REQUIRED true
  !BRIEF A float parameter
!END
!PARAMETER -str
  !TYPE string
  !REQUIRED true
  !BRIEF A string parameter
!END
```

(continues on next page)

(continued from previous page)

```
";

    public static void Main(string [] argv)
    {
        OEInterface itf = new OEInterface(InterfaceData, "UsingOEInterfaceRequired",␣
↪argv);

        Console.WriteLine("-b = {0}", itf.GetBool("-b"));
        Console.WriteLine("-i = {0}", itf.GetInt("-i"));
        Console.WriteLine("-f = {0}", itf.GetFloat("-f"));
        Console.WriteLine("-str = {0}", itf.GetString("-str"));
    }
}
```

If any required parameter is missing from the command line during the command line parsing stage an appropriate error message will be thrown. The following demonstrates the error messages thrown by the program in *Listing 4*:

Legal and Illegal Values and Ranges

Some parameter types can have legal/illegal ranges and values. These are set with the `!LEGAL_VALUE`, `!ILLEGAL_VALUE`, `!LEGAL_RANGE`, and `!ILLEGAL_RANGE` fields. The following are the restrictions for certain parameter types:

bool

Boolean parameters cannot have legal/illegal ranges and values.

int and float

These parameters can have legal/illegal ranges and values. Checks for illegal ranges and values are performed first and if an invalid range or value is detected an error is returned. Then legal ranges and values are checked, and if the parameter setting matches or there are no legal ranges or values it passes. `inf` and `-inf` can be used as range limits, along with regular numerical values.

string

String parameters can have legal/illegal values, but not ranges. Checks for illegal values are performed first and if an invalid value is detected an error is returned. Then legal values are checked, and if the parameter setting matches it passes. Legal/illegal values for the string type support up to 2 * wildcards for pattern matches. This is commonly used to force a file extension, *e.g.*, `!LEGAL_VALUE *.txt`.

Parameter Lists

Sometimes it is useful to allow multiple values for a given parameter. The `!LIST` field is used to specify that a parameter can have multiple values separated by spaces on the command line. The *OEInterface.GetList* method should then be used to get an iterator over the parameter's values. *Listing 5* demonstrates how to specify that a parameter is a list of values and then how to access it with the *OEInterface.GetList* method.

Listing 5: Specifying that a parameter is a list of values

```
using System;
using OpenEye.OEChem;


public class UsingOEInterfaceList
{
```

(continues on next page)

(continued from previous page)

```
    static String InterfaceData = @"
!PARAMETER -strs
  !TYPE string
  !LIST true
  !BRIEF Some string parameters
!END
";

    public static void Main(string [] argv)
    {
        OEInterface itf = new OEInterface(InterfaceData, "UsingOEInterfaceList", argv);

        Console.Write("-strs = ");
        foreach (string param in itf.GetStringList("-strs"))
        {
            Console.WriteLine(param);
            Console.Write("        ");
        }
        Console.WriteLine();
    }
}
```

The command line parser will then assign elements as values based upon their position in the `argv` that was passed to the program. This allows the user to pass values that have spaces in them by simply quoting the relevant section. The following demonstrates the behavior of the program in *Listing 5*:

Positional Parameters

For commonly used parameters it is often useful to allow them to be specified by their position on the command line. The `!KEYLESS` record is used to specify that a parameter can be assigned from a positional value on the command line. *Listing 6* demonstrates how to specify input and output file name parameters that can be specified from their position on the command line.

Listing 6: Specifying keyless input and output file parameters

```
using System;
using OpenEye.OEChem;

public class UsingOEInterfaceKeyless
{
    static String InterfaceData = @"
!PARAMETER -i
  !TYPE string
  !KEYLESS 1
  !BRIEF Input file name
!END
!PARAMETER -o
  !TYPE string
  !KEYLESS 2
  !BRIEF Output file name
!END
";
```

(continues on next page)

(continued from previous page)

```
    public static void Main(string [] argv)
    {
        OEInterface itf = new OEInterface(InterfaceData, "UsingOEInterfaceKeyless",␣
↪argv);

        Console.WriteLine("Input file name = " + itf.GetString("-i"));
        Console.WriteLine("Output file name = " + itf.GetString("-o"));
    }
}
```

!KEYLESS parameters can still be specified using their command line flag. However, positional arguments need to be the last values on the command line. The following demonstrates how !KEYLESS parameters can and cannot be specified on the command line:

Warning

An ambiguity occurs when mixing !KEYLESS with !LIST. There is no way for the parser to differentiate between multiple values for a !LIST and different !KEYLESS parameters. *OEInterface* resolves this by requiring that any keyless list parameter must be the *last* keyless parameter.

Molecule Parameters

As the previous example illustrated how to handle simple parameters such as `float`, `int` or `string`. The *OEInterface* also provides a convenient way to open a molecule file and return its first molecule.

Similarly the first multi-conformer molecule can be imported from a molecule file by setting the type of a parameter as !TYPE OEMol.

Command Line Help

The *OEInterface* object can have a significant amount of information, default values, text descriptions, etc., about the parameters it holds. All this information allows the *OEInterface* to provide help to the user interactively automatically based upon the interface definition. *Listing 8* is a fairly simple program, it just concatenates all the molecules given into a single molecule. Though it has a fairly complex set of command line parameters which *OEInterface* handles automatically.

Listing 8: A program to concatenate any number of molecules together

```
using System;
using OpenEye.OEChem;

public class UsingOEInterfaceHelp
{
    static String InterfaceData = @"
!BRIEF UsingOEInterfaceHelp [-d <delimiter>] [-o] <output> [-i] <input1> <input2> ...
!PARAMETER -delim
  !ALIAS -d
  !TYPE string
  !DEFAULT _
  !BRIEF Title delimiter
  !DETAIL
```

(continues on next page)

(continued from previous page)

```
        This is the value given to the OEChem function OEAddMols to
        separate the titles of the input molecules in the output.
!END
!PARAMETER -output
  !ALIAS -o
  !TYPE string
  !REQUIRED true
  !KEYLESS 1
  !BRIEF The output file
  !DETAIL
        The molecule file to output to, can be any file format
        OEChem supports.
!END
!PARAMETER -inputs
  !ALIAS -i
  !TYPE string
  !REQUIRED true
  !LIST true
  !KEYLESS 2
  !BRIEF The input files
  !DETAIL
        A list of molecule files to add together. Note, only the
        first molecule from every file will be added.
!END
";

    public static void Main(string [] argv)
    {
        OEInterface itf = new OEInterface(InterfaceData, "UsingOEInterfaceHelp", argv);

        OEGraphMol outmol = new OEGraphMol();
        foreach (string param in itf.GetStringList("-inputs"))
        {
            oemolistream ims = new oemolistream(param);

            OEGraphMol inmol = new OEGraphMol();
            OEChem.OEReadMolecule(ims, inmol);

            OEChem.OEAddMols(outmol, inmol, itf.GetString("-delim"));
        }

        oemolostream oms = new oemolostream(itf.GetString("-output"));
        OEChem.OEWriteMolecule(oms, outmol);
    }
}
```

Note

- `!BRIEF` can be used outside of a `!PARAMETER` to give a concise one line command line usage.
- `!ALIAS` is used to specify an alternative name the parameter can be specified with.

- `!DETAIL` is used for a more detailed description of the parameter.

Whenever there are `!REQUIRED` parameters the *OEInterface* will complain about an empty command line. This is usually the first way users interact with a command line program so the goal is to give them only enough information to get the program running. This is where specifying a top-level `!BRIEF` is exceptionally useful to the end user. The following is the output of the program in *Listing 8* when no parameters are given:

More information is accessible from the command line by specifying `--help` as the first argument on the command line. The following is an example of requesting help from the program in *Listing 7*:

As seen above `--help` can take arguments that allow the user introspect even further into the semantics of a particular parameter.

Note

OECheckHelp is the low-level function used to check a command line for `--help`. By default the *OEInterface constructor* and the *OEParseCommandLine* function call *OECheckHelp* automatically so most users will never need to use this function directly.

Interface Definition Specification

Configuring an *OEInterface* is the process of telling the name, type, and other details of all the command line parameters the program is going to use. An interface definition consists of a collection of `PARAMETER` blocks optionally organized into `CATEGORY` blocks. A `PARAMETER` block must always have the following information:

NAME
The command line flag, *e.g.*, `-i`, `-str`, `-cutoff`

TYPE
The kind of information being passed, *e.g.*, `float`, `int`, `string`, `bool`

A `PARAMETER` can also optionally contain the following additional information:

ALIAS
One or more aliases for the parameter name that are generally used to create shorthand for lengthy parameter names. For example a parameter named `-energy_cutoff_window` could be given an alias `-ecw`, that would make typing `-energy_cutoff_window 10.0` on the command line equivalent to `-ecw 10.0`.

DEFAULT
A default value for the parameter that will be used if the parameter is not specified by the user on the command line.

REQUIRED
A flag specifying that the parameter is required. If a parameter is required the program will throw an error if the user does not specify that parameter. By default parameters are not required.

LIST
A flag specifying whether the parameter is a list of multiple values. For example, `./program -inputs file1.smi file2.smi file3.smi`.

KEYLESS
A number, `N`, that specifies that this parameter may be fulfilled by the `Nth` value on the command line that does not have a parameter flag specifically associated with it. For example, if `-i` had a `KEYLESS` value of `1` specified the following are equivalent command lines, `./program -i foo.smi` and `./program foo.smi`.

VISIBILITY
A visibility level that controls when the parameter appears when the users requests `--help`. Valid values are: `simple`, `normal`, `hidden`. The default is `simple` if a `VISIBILITY` isn't specified.

LEGAL_VALUE
A valid value that the parameter can be. The parameter specified must be one of the values specified in a `LEGAL_VALUE` field.

ILLEGAL_VALUE
A value that cannot be specified as the parameter value.

LEGAL_RANGE
A range that the parameter value must fall into. More useful than `LEGAL_VALUE` when the parameter is a `float`.

ILLEGAL_RANGE
A range that the parameter value must not fall into. More useful than `ILLEGAL_VALUE` when the parameter is a `float`.

BRIEF
A one line description of the parameter that will be displayed to users when listing multiple command line options using the high-level `--help` functions.

DETAIL
A multi line description of the parameter that will be displayed to users when the user requests help about a specific parameter, *e.g.*, `--help -foo`.

Therefore, a `PARAMETER` block takes the following form:

```
!PARAMETER <name> [order priority]
  !ALIAS <alias>
  !TYPE <type>
  !DEFAULT <default value>
  !REQUIRED <true or false>
  !LIST <true or false>
  !KEYLESS <N>
  !VISIBILITY <visibility>
  !LEGAL_VALUE <value>
  !ILLEGAL_VALUE <value>
  !LEGAL_RANGE <hi value> <low value>
  !ILLEGAL_RANGE <hi_value> <low_value>
  !BRIEF <brief description>
  !DETAIL
    <detailed description line 1>
    <detailed description line 2>
    <detailed description line 3>
    .
    .
    .
!END
```

`PARAMETER` blocks can then be organized into `CATEGORY` blocks. `CATEGORY` blocks can also contain nested `CATEGORY` blocks, there is no limit to the amount of nesting. The value specified after the `!CATEGORY` specifier is the name of the `CATEGORY`. The `CATEGORY` can also contain a `!BRIEF` to give a human readable description of the grouping. The following interface definition defines an interface definition with an "Input and Output" category with sub-categories for input and output respectively. As well as a category for various other parameters.

```
!CATEGORY Input and Output
  !CATEGORY Input
    !PARAMETER -i
      ...
    !END
  !CATEGORY Output
    !PARAMETER -o
      ...
    !END
  !END
!END
!CATEGORY Other Stuff
  !PARAMETER -o
    ...
  !END
!END
```

By default parameters and categories appear in alphabetical order. This order can be adjusted however by adding an integer after the category or parameter name in the interface definition. Parameters and categories are then sorted first by this integer and secondarily by their names. If this integer is unspecified, it is assumed to be zero. The following example illustrates re-ordering of both categories and variables.

```
!CATEGORY Input and Output 1
  !CATEGORY Input 1
    !PARAMETER -input 1
      ...
    !END
    !PARAMETER -dbase 2
      ...
    !END
 !CATEGORY Output 2
    !PARAMETER -output 1
      ...
    !END
    !PARAMETER -hits 2
      ...
    !END
  !END
!END
!CATEGORY Other Stuff 2
  !PARAMETER -o
    ...
  !END
!END
```

2.2.3 Generic Data

Previous chapters *Molecule Properties*, *Atom Properties*, *Bond Properties* have described how common global properties of molecule, atoms, and bonds can be modified and accessed. There are applications, however, when associating arbitrary data with objects such as molecules, atoms and bonds is necessary. **OEChem TK** provides a framework to solve this problem by allowing to attach **generic data** to an object by association either with an integer or character string, called **tag** identifier.

The following two snippets demonstrate how generic data (for example molecule weight) can be attached to a molecule:

```
uint tag = OEChem.OEGetTag("MolWeight");
mol.SetDoubleData(tag, OEChem.OECalculateMolecularWeight(mol));
```

```
mol.SetDoubleData("MolWeight", OEChem.OECalculateMolecularWeight(mol));
```

After annotation, the data can be accessed with the same integer or character string identifier:

```
tag = OEChem.OEGetTag("MolWeight");
Console.WriteLine(mol.GetDoubleData(tag));
```

```
Console.WriteLine(mol.GetDoubleData("MolWeight"));
```

Warning

The integer tag of a generic data should always be allocated using the *OEGetTag* function.

The following table shows the basic methods of the *OEBase* class that allow the manipulation of generic data.

Table 19: **Methods to manipulate generic data**

Method	Description
OEBase.SetData	sets a generic data associating it with the given tag
OEBase.AddData	adds a generic data associating it with the given tag
OEBase.HasData	determines whether a molecule has any generic data with a given tag
OEBase.GetData	returns the generic data associated with the given tag
OEBase.DeleteData	deletes all generic data with the given tag
OEBase.Clear	clears all stored generic data

The main difference between the *OEBase.SetData* method and the *OEBase.AddData* method is that if a data with the same identifier is already attached to an object then:

- by using *OEBase.SetData*, the data is replaced
- by using *OEBase.AddData*, the data is appended

Furthermore, *OEBase.SetData* does not allow replacing an existing tag with a different data type:

```
uint tag = OEChem.OEGetTag("MolWeight");
double weight =  OEChem.OECalculateMolecularWeight(mol);
mol.SetDoubleData(tag, weight);
mol.SetIntData(tag, (int)weight);
```

The above code will throw the following warning:

```
Warning: data type mismatch found when using generic data
```

Attaching plain old data

The following simple code demonstrate how data calculated and attached to a molecule in one function can be accessed later on through the tag identifier.

Listing 1: Example of using generic data

```
using System;

using OpenEye.OEChem;

public class GenericDataMolWeight
{
    private static void CalculateMoleculeWeight(OEMolBase mol)
    {
        mol.SetDoubleData("MolWeight", OEChem.OECalculateMolecularWeight(mol));
    }

    private static void PrintMoleculeWeight(OEMolBase mol)
    {
        uint tag = OEChem.OEGetTag("MolWeight");
        if (mol.HasData(tag))
        {
            Console.WriteLine("molecule weight = {0}", mol.GetDoubleData(tag));
        }
        else
        {
            Console.WriteLine("molecule weight is not calculated!");
        }
    }

    public static int Main(String[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "C1CCCC(C(=O)O)C1");

        CalculateMoleculeWeight(mol);
        PrintMoleculeWeight(mol);
        return 0;
    }
}
```

> **Note**
>
> It is a good programming practice to call *OEBase.HasData* in order to check whether or not a data exists before trying to access it by the *OEBase.GetData* method.

Attaching data to atoms

Generic data can be attached to any object that derives from the *OEBase* class. The following program shows an example where hydrogen bonding donor property is attached as a bool value to the corresponding *OEAtomBase* object.

Listing 2: Example of attaching generic data to atoms

```
using System;

using OpenEye.OEChem;

public class GenericDataToAtom
{
    private class IsDonorAtomPred : OEUnaryAtomPred
    {
        public override bool Call(OEAtomBase atom)
        {
            return atom.GetBoolData("isdonor");
        }
        public override OEUnaryAtomBoolFunc CreateCopy()
        {
            OEUnaryAtomBoolFunc copy = new IsDonorAtomPred();
            copy.ReleaseOwnership();
            return copy;
        }
    }

    public static void Main(String[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1c(Cl)cncc1C(=O)O");

        OEMatchAtom IsDonorAtom = new OEMatchAtom("[!H0;#7,#8]");
        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            atom.SetBoolData("isdonor", IsDonorAtom.Call(atom));
        }

        Console.Write("Donor atoms: ");
        foreach (OEAtomBase atom in mol.GetAtoms(new IsDonorAtomPred()))
        {
            Console.Write(atom.GetIdx() + " " +  OEChem.OEGetAtomicSymbol(atom.
↪GetAtomicNum()));
        }
        Console.WriteLine("");
    }
}
```

See also

- *OEBase* class in the *OESystem Classes* chapter for the full list of derived classes.

Attaching other objects

The type of the generic data is not restricted to fundamental data types of the programming language. High-level **OEChem TK** objects such as *OEMolBase*, *OEAtomBase*, *OEBondBase OEScalarGrid*, *OESkewGrid* and OESurface can also be stored through this mechanism. The following program demonstrates how to attach a subset of a molecule to the original molecule as generic data.

Listing 3: Example of attaching a molecule as generic data

```
using System;

using OpenEye.OEChem;

public class GenericDataMol
{
    public static void Main(String[] args)
    {
        OEGraphMol mol = new OEGraphMol();
        OEChem.OESmilesToMol(mol, "c1ccccc10");

        OEGraphMol frag = new OEGraphMol();
        OEChem.OESubsetMol(frag, mol, new OEIsCarbon());

        OEChem.OESetOEGraphMol(mol, "just_carbon", frag);
        OEGraphMol justCarbon = OEChem.OEGetOEGraphMol(mol, "just_carbon");
    }
}
```

Note

Generic data attached to a molecule or any of its atoms or bonds is automatically saved when the molecule is written into an `.oeb` file.

See also

- *SD Tagged Data Manipulation* section
- *PDB Tagged Data Manipulation* section

2.2.4 OERecord

An *OERecord* is a data container that holds a variety of strongly-typed, named values, plus associated metadata. It is used to pass data between cubes in Orion workfloes, and is the main mechanism by which data is communicated to and from Orion. OERecords also provide a general way to create and store structured data within the **OpenEye** ecosystem.

The most straightforward use of an *OERecord* is for storage and retrieval of simple values. The following example illustrates how to use a record to store and retrieve integer, string, and floating point values.

OERecord has methods that allow you to interrogate the record for the presence of fields and values.

Other types of data can be stored on OERecords, including molecules and even other OERecords.

OERecords can be written to files and read from files using the OEWriteRecord and *OEReadRecord* functions.

Metadata on fields

There are two types of metadata: *options* which are simple flags, and *attributes* which are key/value pairs with user-supplied values.

In the following example, two pieces of metadata, one option and one attribute, are put on a field to describe the contents of that field.

Metadata can also be used to designate relationships between fields on a record. The following example shows how to attach error values to a field using metadata.

See also

- *OEMolRecord* chapter

2.3 OEBio Theory

Working With Macromolecules

2.3.1 Biopolymers

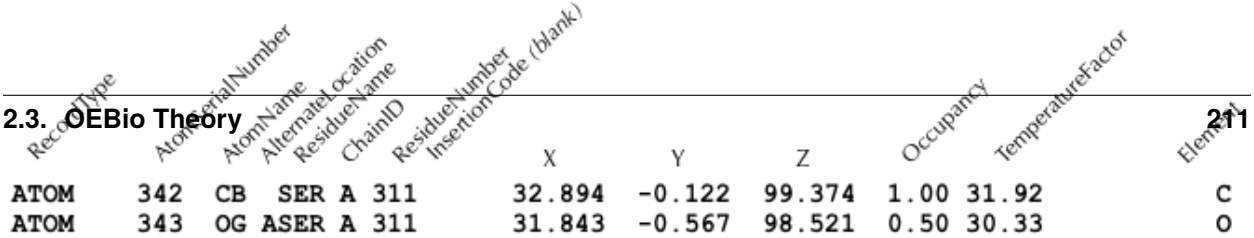
Biopolymer Residues

Living organisms manufacture an assortment of biopolymers: proteins are polymers of amino-acid residues, DNA and RNA are polymers of nucleotide bases, and polysaccharides such as cellulose and starch are polymers of sugars. In a description of these often very large molecules, it is useful to include with each atom information about the monomer it resides in, the chain the monomer is in, the model the chain is in, etc. **OEChem TK**'s `OEResidue` is a container for just such information, and more.

Table 20: Properties of OEResidues

Property	Type	Get Method	Set Method
Residue Name	char*	`GetName`	`SetName`
Residue Number	int	`GetResidueNumber`	`SetResidueNumber`
Insertion Code	char	`GetInsertCode`	`SetInsertCode`
Chain ID	char	`GetChainID`	`SetChainID`
NMR Model Number	int	`GetModelNumber`	`SetModelNumber`
Fragment Number	int	`GetFragmentNumber`	`SetFragmentNumber`
Secondary Structure	int	`GetSecondaryStructure`	`SetSecondaryStructure`
Heterogen	bool	`IsHetAtom`	`SetHetAtom`
Atom Serial Number	int	`GetSerialNumber`	`SetSerialNumber`
Alternate Location Code	char	`GetAlternateLocation`	`SetAlternateLocation`
Occupancy	float	`GetOccupancy`	`SetOccupancy`
Temperature Factor	float	`GetBFactor`	`SetBFactor`

Note that `OEResidue`



is actually a part of **OEChem TK**, it is included in this **OEBio TK** chapter because it underlies much of **OEBio TK**'s functionality. *Table: Properties of OEResidues* lists each property along with the *Get* and *Set* methods for each. *Figure: PDB format atom records* shows how these properties are encoded for a structure in PDB format.

The *residue number*, *insertion code*, *chain ID*, *model number* and *fragment number* properties describe a hierarchy of structural elements, but do not impose any explicit hierarchy. Because the biological world is inherently messy, the flexibility of a flat representation such as this is very useful. For example, because it does not require a chain to be either *above* or *below* a fragment, it can describe both a chain with gaps, composed of multiple covalently bonded fragments and one that is covalently bonded to another chain. And, although residues are in a given sequence order, it does not require the polymer to be linear and can easily handle a glycosylated protein where a chain may have a tree-like, rather than linear, structure.

On the other hand, for some tasks a hierarchical view of biopolymers is more natural and more efficient, despite the limitations of a hierarchy. For more information, see the section *A Hierarchy View*.

Note that although infrequently used, the *insertion code* modifies the *residue number*. When comparing residue numbers it is necessary to compare *both* the residue number and the insertion code. The best way to avoid errors when comparing OEResidues is to use *OESameResidue*.

Listing 1: Accessing OEResidue properties

```
using System;
using OpenEye.OEChem;

public class SimpleResCountDoc
{
    public static void Main(string[] argv)
    {
        if (argv.Length != 1)
        {
            OEChem.OEThrow.Usage("SimpleResCount <mol-infile>");
        }

        oemolistream ims = new oemolistream();
        if (! ims.open(argv[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + argv[0] + " for reading");
        }

        OEGraphMol mol = new OEGraphMol();

        while (OEChem.OEReadMolecule(ims, mol))
        {
```

(continues on next page)

(continued from previous page)

```
            CalcResCounts(mol);
        }
    }

    private static void CalcResCounts(OEMolBase mol)
    {
        if (! OEChem.OEHasResidues(mol))
            OEChem.OEPerceiveResidues(mol, OEPreserveResInfo.All);

        int resCt = 0;
        int hetCt = 0;
        OEResidue prevRes = new OEResidue();

        foreach (OEAtomBase atom in mol.GetAtoms())
        {
            OEResidue thisRes = OEChem.OEAtomGetResidue(atom);
            if (!OEChem.OESameResidue(prevRes, thisRes))
            {
                ++resCt;
                if (thisRes.IsHetAtom())
                {
                    ++hetCt;
                }

                prevRes = thisRes;
            }
        }

        Console.WriteLine("Molecule: " + mol.GetTitle());
        Console.WriteLine("Residues: " + resCt + " (" + hetCt + " hets)");
    }
}
```

It is important to keep in mind that *OEResidues* are not *residues*, but instead are information blocks associated with atoms. Several of the properties of an *OEResidue* such as *atom serial number* and *occupancy* are not residue properties at all, but instead are properties of a specific atom. Changing an *OEResidue* property will not modify anything in the molecule until the *OEResidue* is saved back into the molecule, and this must be done for each atom for which the property is to be changed.

Listing 2: Updating OEResidue properties

```
foreach (OEAtomBase atom in mol.GetAtoms())
{
    OEResidue thisRes = OEChem.OEAtomGetResidue(atom);
    if (OEChem.OEGetResidueIndex(thisRes) == OEResidueIndex.MSE)
    {
        thisRes.SetName("MET");            // modify res properties
        thisRes.SetHetAtom(false);
        OEChem.OEAtomSetResidue(atom, thisRes); // store updated residue

        if (atom.GetAtomicNum() == OEElemNo.Se)
        {
```

(continues on next page)

(continued from previous page)

```
            atom.SetAtomicNum(OEElemNo.S); // fix atom type & name
            atom.SetName(" SD ");
        }
    }
}
```

The input routines for Protein Data Bank (PDB) files are setup to automatically perceive *OEResidue* properties but the perception can also be done explicitly, if required, as shown below.

Listing 3: Perceiving OEResidues

```
if (!OEChem.OEHasResidues(mol))
{
    OEChem.OEPerceiveResidues(mol, OEPreserveResInfo.All);
}
```

Residue Naming Conventions

The standard residue name list is ALA, ARG, ASN, ASP, CYS, GLN, GLU, GLY, HIS, ILE, LEU, LYS, MET, PHE, PRO, SER, THR, TRP, TYR, VAL, ASX, GLX, CYX, CYH, HID, HIE or HIP.

GLX and ASX are the Protein Data Bank's standard names for when it is not clear whether the residue is a GLU or GLN, or an ASP or ASN.

CYH and CYX are ways that groups outside of the PDB describe explicitly reduced and oxidized version of CYS, either cysteine or cystine respectively.

We generally follow the standard-non-standard names from the AMBER force field. For example, we show ASH, CYS, CYH, CYM, GLH, HID, HIE, HIP, LYN, TYM, where H at the end means reduced, N means neutral and M means deprotonated. We will not generate PDB files or rename residues in molecules based on their physical state, but will follow the standard residue names like HIS to cover all states of HIE, HID, HIP. We also respect residue names provided to us.

We discourage the use of UNX, UNK and UNL according to: Worldwide PDB Heterogen Section. They should only be used in instances where the chemical components for which the chemical identity is truly unknown. From the wwPDB: Unknown atoms or ions will be represented as UNX with the chemical formula X1. Unknown ligands are UNL; unknown amino acids are UNK.

A Hierarchy View

At times, it may be more convenient to loop over residues or chains directly, rather than looping over atoms and keeping track as *OEResidue* properties change. The *OEHierView* class supports this approach by representing a biopolymer molecule as a hierarchy of the components: *OEHierChain*, *OEHierFragment*, and *OEHierResidue*.

Listing 4: Looping over hierarchical components

```
using System;
using OpenEye.OEBio;
using OpenEye.OEChem;

public class ResCountDoc
{
    public static void CalcResCounts(OEMolBase mol)
    {
```

(continues on next page)

(continued from previous page)

```
        OEHierView hv = new OEHierView(mol);
        uint chainCt = 0;
        uint fragCt  = 0;
        uint resCt   = 0;
        uint watCt   = 0;
        foreach (OEHierChain chain in hv.GetChains())
        {
            ++chainCt;
            foreach (OEHierFragment frag in chain.GetFragments())
            {
                ++fragCt;
                foreach (OEHierResidue hres in frag.GetResidues())
                {
                    ++resCt;
                    if (OEChem.OEGetResidueIndex(hres.GetOEResidue())
                                                        == OEResidueIndex.HOH)
                    {
                        ++watCt;
                    }
                }
            }
        }
        Console.WriteLine("Molecule : " + mol.GetTitle());
        Console.WriteLine("Chains   : " + chainCt);
        Console.WriteLine("Fragments: " + fragCt);
        Console.WriteLine("Residues : " + resCt   + " (" + watCt + " waters)");
    }

    public static int Main(string[] args)
    {
        if (args.Length != 1)
        {
            OEChem.OEThrow.Usage("ResCountDoc <mol-infile>");
        }

        oemolistream ims = new oemolistream();
        if (!ims.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ims, mol))
        {
            if (! OEChem.OEHasResidues(mol))
                OEChem.OEPerceiveResidues(mol, OEPreserveResInfo.All);

            CalcResCounts(mol);
        }
        ims.close();
        return 0;
    }
```

(continues on next page)

(continued from previous page)

```
}
```

Be aware that *OEHierView* works with what amounts to a snapshot of the molecule, so if changes are made to the original molecule (such as adding or deleting atoms) it may be necessary to rebuild the *OEHierView*.

The *OEHier...* classes provide direct access to chains, fragments and residues using common identifiers such as *chain id*, *residue name* and *residue number*.

Listing 5: Selecting a residue

```
OEHierView hv = new OEHierView(mol);
OEHierResidue hres = hv.GetResidue('A', "LEU", 27);
foreach (OEAtomBase atom in hres.GetAtoms())
{
    OEResidue res = OEChem.OEAtomGetResidue(atom);
    Console.WriteLine(res.GetSerialNumber() + " " + atom.GetName());
}
```

Sequence Alignment

Two proteins can be sequence aligned with the function *OEGetAlignment* which returns the class *OESequenceAlignment*. Several different amino acid distance matrices are supplied, including *OESeqAlignmentMethod.PAM250* [Dayhoff-1978] for closely related proteins, and *OESeqAlignmentMethod.BLOSUM62* [Henikoff-1992] & *OESeqAlignmentMethod.GONNET* [Gonnet-1992] for more distantly related proteins.

Alignments can be written out in standard format using the function *OEWriteAlignment*.

The function *OERMSD* will calculate the backbone or C-α RMSD of previously aligned proteins.

Crystal Symmetry

The space group and unit cell parameters (a, b, c, α, β, γ) for an x-ray crystal structure can be accessed and modified using the following *OEBio* functions:

- *OEHasCrystalSymmetry*
- *OEGetCrystalSymmetry*
- *OESetCrystalSymmetry*
- *OECopyCrystalSymmetry*
- *OEGetSpaceGroupNumber*
- *OEGetSpaceGroup*

The function *OEExpandCrystalSymmetry* will replicate the molecule to fill-in each of the symmetry related molecules in other regions of the crystal lattice, out to a specified radius.

Protein Secondary Structure

OEBio provides functionality for perceiving secondary structural elements in proteins. Perception is performed according to the method of Kabsch and Sander [Kabsch-1983] and identifies helices, beta sheets, and turns based on hydrogen bonding patterns within a protein. The function *OEPerceiveSecondaryStructure* determines secondary structure for a protein molecule, and stores the assigned secondary structure types in the *OEResidue* associated with each atom in the protein. Secondary structure assignments are stored as single integer values within *OEResidue*, into which are packed the type (helix, sheet, etc.), helix/sheet/turn ID numbers, and strand identifiers for individual strands

within beta sheets. To help decode the secondary structure information packed in these integers, several functions are provided:

- *OEGetSecondaryStructureType*
- *OEGetSecondaryStructureID*
- *OEGetSecondaryStructureStrandID*
- *OEGetSecondaryStructureName*

These functions all take an integer argument, which is typically the value returned from *OEResidue.GetSecondaryStructure*. An additional function, *OESecondaryStructurePacked*, will do the reverse operation, namely, taking a secondary structure type, ID, and optional strand ID, and packing them into an integer suitable for use with *OEResidue.SetSecondaryStructure*.

Secondary structure types are defined in the *OESecondaryStructure* namespace.

- *Unassigned*
- *Helix*
- *Helix310*
- *HelixAlpha*
- *HelixPi*
- *Turn*
- *Turn3*
- *Turn4*
- *Turn5*
- *Sheet*
- *StrandSense*
- *HelixLeft*
- *Any*

A functor is also provided to help identify atoms belonging to particular secondary structure features. *OEHasSecondaryStructure* takes an integer argument. This integer can be one of the above constants (e.g., to find all atoms belonging to alpha helices); the value retrieved from a call to *OEResidue.GetSecondaryStructure* (e.g., to find all atoms belonging to the same helix as a particular residue); or a value returned from *OESecondaryStructurePacked*, which can be used to find atoms from a specific secondary structural element, such as Helix number 4.

Functors

A number of atom predicates are provided in *OEBio* to make it easy to identify atoms with particular properties:

- *OEAtomMatchResidue*
- *OEIsAlphaCarbon*
- *OEIsBackboneAtom*
- *OEIsWater*
- *OEHasAlternateLocation*
- *OEHasSecondaryStructure*
- *OEHasPDBAtomIndex*

This last predicate uses PDBAtoms (such as *CA*) to refer to specific atom types in a Protein Data Bank (PDB) structure.

Misc

The function *OEIsStandardProteinResidue* indicates whether a given *OEHierResidue* is a standard amino acid.

The function *OEHasBondedResidues* indicates whether the atoms in each residue have bonds to the same residue.

Given an *OEAtomBase* or *OEResidue*, the function *OEGetResidueAtoms* will return an iterator over all the atoms in the residue.

2.3.2 Protein Preparation

Protein preparation is the process of transforming a macromolecular structure into a form more suitable for a computational experiment. The specific set of tasks required depends on the experiment and the source of the structure. Tools for several common prep tasks are described below. For a demonstration of how these can be combined, see the *Preparing a Protein* example.

For a more comprehensive protein preparation process, Advanced Macromolecular Preparation Workflow shows an end-to-end preparation workflow to process protein structures for molecular modeling. For more information on **Spruce TK** see the associated Theory and Spruce API.

Processing Alternate Locations

Protein structures rarely describe whole molecules. Instead, they are usually missing atoms that could not be resolved well. In addition, many structures contain *multiple* copies of atoms called alternate ("alt") locations or alternate conformations that can indicate that part of the structure is moving. As the resolution of X-ray crystal structures improves, alt locations can be expected to become ubiquitous. These conformations are indexed by an ID character such as `A` or `B` that is stored in the residue information for that atom (see *Biopolymer Residues*).

Having extra copies of atoms can interfere with some experiments, such as docking. One way to handle this is to eliminate all but one conformation. By default, **OEChem TK** drops all but the `A` or *blank* conformations when reading atoms in a PDB format file. To prevent this, the file must be read with the *OEIFlavor.PDB.ALTLOC* flavor.

Listing 1: Setting the PDB Flavor to Retain All Alts

```
oemolistream ims = new oemolistream();
ims.SetFlavor(OEFormat.PDB, OEIFlavor.PDB.ALTLOC);
```

Another way to drop all but one conformation is to use the *OEAltLocationFactory* object (see the *Alternate Locations* section) and generate, for example, a molecule that retains only the highest occupancy conformation of each location group. This object can be used to iterate through each combination of alt locations (for example, when a group is moving in an active site).

Listing 2: Keeping the Highest Occupancy Alt Locations

```
OEAltLocationFactory alf = new OEAltLocationFactory(mol);
if (alf.GetGroupCount() != 0)
    alf.MakePrimaryAltMol(mol);
```

Adding Hydrogens and Optimizing Hydrogen Bonds

Traditionally, most protein structures have been generated with either no explicit hydrogens or only polar hydrogens. However, having all atoms explicitly represented and positioned to make appropriate hydrogen bonds is sometimes necessary. The function *OEPlaceHydrogens* does this by adding and placing hydrogens and by "flipping" certain functional groups (e.g., sidechain amides and imidazoles) as required.

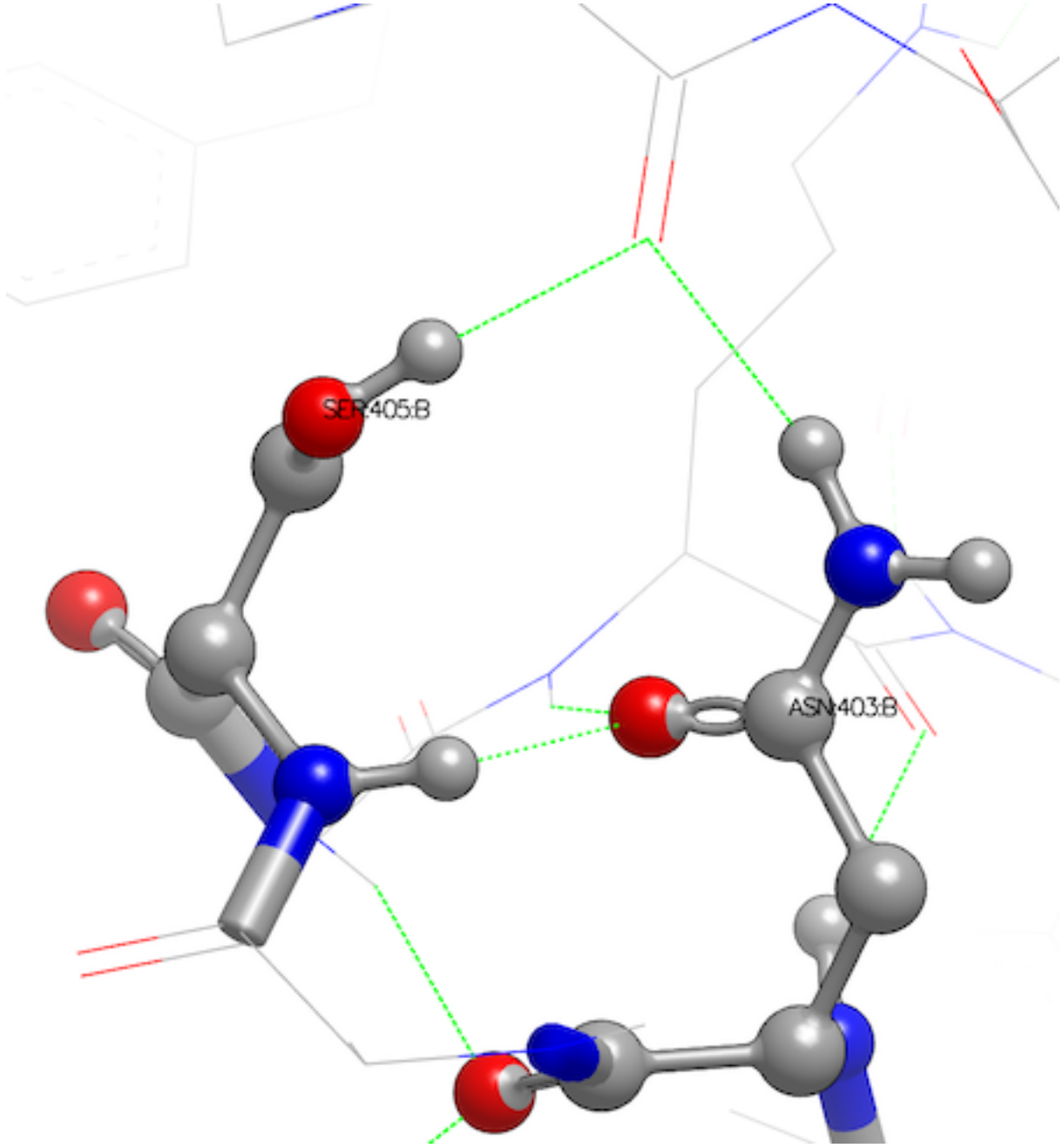


Fig. 79: **Asparagine has been flipped and serine oriented to H-bond with main-chain atoms**

Listing 3: Adding and Optimizing the Positions of Hydrogens

```
if (OEBio.OEPlaceHydrogens(mol))
{
    // ...
```

There are several *OEPlaceHydrogensOptions* parameters that control the process. For example, it is possible to prevent the bond lengths of any covalent bonds to pre-existing hydrogens from being altered:

Listing 4: Setting Options for Placing Hydrogens

```
OEPlaceHydrogensOptions opt = new OEPlaceHydrogensOptions();
opt.SetStandardizeBondLen(false);
```

There are also *OEPlaceHydrogensOptions* parameters for restraining which states a mover group is sampled over. *Listing 5* shows a predicate for identifying atoms belonging to a specific residue.

Listing 5: Predicate for Residue Equivalence

```
private class IsSameResidue : OEUnaryAtomPred
{
    private OEResidue referenceResidue;

    public IsSameResidue(OEResidue res)
    {
        referenceResidue = new OEResidue(res);
    }

    public override bool Call(OEAtomBase atom)
    {
        OEResidue res = OEChem.OEAtomGetResidue(atom);
        return OEChem.OESameResidue(res, referenceResidue);
    }

    public override OEUnaryAtomBoolFunc CreateCopy()
    {
        OEUnaryAtomBoolFunc copy = new IsSameResidue(referenceResidue);
        copy.ReleaseOwnership();
        return copy;
    }
}
```

We can use that predicate to prevent a selected residue from flipping:

Listing 6: Preventing a Residue from Flipping

```
// given predicate IsSameResidue, residue selectedResidue and molecule mol...
OEPlaceHydrogensOptions opt = new OEPlaceHydrogensOptions();
opt.SetNoFlipPredicate(new IsSameResidue(selectedResidue));

if (OEBio.OEPlaceHydrogens(mol, opt))
{
    // selectedResidue will not be flipped...
```

Using an *OEPlaceHydrogensDetails* object enables examination of the results, showing the clusters of movable groups explored and the scores. A simple way to get at this information is to display a summary description:

Listing 7: Examining the Details of Placing Hydrogens

```
// given molecule mol and OEPlaceHydrogensOptions opt...
OEPlaceHydrogensDetails details = new OEPlaceHydrogensDetails();
if (OEBio.OEPlaceHydrogens(mol, details, opt))
    Console.WriteLine(details.Describe());
```

The following is a fragment from a details object description:

```
OEBio 2.1.1: score system MMFF-NIE: flip bias 2.000
87 clusters : 5 flips
cluster 0  : score -53.935!
             CG  ASN44(A)   :      amide:   -6.084  (o=-53.93!,f=-8.42!)
             NZ  LYS47(A)   : NH3 150deg:  -14.865
             CG  HIS80(A)   :    +bothHN:   -5.628  (o=-53.93!,f=-15.89!)
             OG1 THR81(A)   : OH   65deg:    1.255
             CD  GLN87(A)   :FLIP  amide:   -4.976  (o=-26.48!,f=-53.93!)
             CG  ASN126(A)  :      amide:   -5.193  (o=-53.93!,f=-35.33!)
             CG  ASN169(A)  :      amide:   -2.230  (o=-53.93!,f=-33.02!)
             CD  GLN203(A)  :      amide:   -1.400  (o=-53.93!,f=-15.53!)
             CG  HIS205(A)  :    +bothHN:   -8.693! (o=-53.93!,f=-6.89!)
             OG1 THR232(A)  : OH  179deg:    9.120
             O3B XYP601(A)  : OH  280deg:   -5.020
             O2B XYP601(A)  : OH  173deg:    3.917
             O4B XYP601(A)  : OH   50deg:    1.236
             O3  XIF602(A)  : OH   29deg:   -3.688
cluster 1  : score -85.724!
             CG  HIS85(A)   :    +bothHN:   -7.737! (o=-85.72!,f=-84.85!)
             OG  SER86(A)   : OH  241deg:    2.116
             OG  SER139(A)  : OH  185deg:    3.367!
             OH  TYR171(A)  : OH   25deg:   -3.384
             CG  ASN172(A)  :      amide:    1.153  (o=-85.72!,f=-68.75!)
             NZ  LYS179(A)  : NH3 188deg:  -39.497
...
single  13 : OG1 THR2(A)    : OH  151deg:    1.545
single  14 : OG  SER25(A)   : OH  292deg:    2.046
single  15 : OH  TYR29(A)   : OH  199deg:   -1.262
single  16 : OG  SER35(A)   : OH  254deg:    1.935
...
```

See also

- the *OEPlaceHydrogensDetails.Describe* method for more details about the output above and the MMFF-NIE scoring system that generated it.

Splitting Macromolecular Complexes

One common step in preparing a macromolecular complex for use as input to a calculation is to separate components based on their functional roles. For example, a **SZMAP** calculation on a four-chain protein crystal structure might require one of four copies of the ligand to be extracted and the protein-complex associated with that ligand binding site to be separately extracted, ignoring any waters, buffers, etc.

Listing 8 uses the *OESplitMolComplex* function to separate the input molecule into the ligand for the first binding site, the protein-complex (including cofactors) for that site, the binding site waters, and everything else. In this example, the ligand is identified automatically and the binding site contains components near the ligand. If a ligand cannot be located, the entirety of each protein and everything nearby defines each binding site. If the input molecule is empty or there is some other problem that prevents processing, *OESplitMolComplex* will return `false`.

Listing 8: Splitting an Input Molecule

```
// for input molecule inmol ...
OEGraphMol lig   = new OEGraphMol();
OEGraphMol prot  = new OEGraphMol();
OEGraphMol wat   = new OEGraphMol();
OEGraphMol other = new OEGraphMol();

if (OEBio.OESplitMolComplex(lig, prot, wat, other, inmol))
    //work with the output molecules lig, prot, ...
```

Listing 8 uses the defaults, so no attempt would be made to recognize covalent ligands. *Listing 9* uses an option to turn on the splitting of covalent ligands. There are a large number of options to control the process of splitting a molecule. See *OESplitMolComplexOptions* for a complete list.

Listing 9: Splitting Covalent Ligands

```
// given input and output molecules ...
OESplitMolComplexOptions opt = new OESplitMolComplexOptions();
opt.SetSplitCovalent();

if (OEBio.OESplitMolComplex(lig, prot, wat, other, inmol, opt))
    // ...
```

A related approach uses *OEGetMolComplexComponents* to return an iterator of molecules. *Listing 10* also illustrates providing an explicit ligand residue name that must match for the ligand to be identified.

Listing 10: Iterating Over Split Components

```
// for input molecule mol and string ligName ...
OESplitMolComplexOptions opt = new OESplitMolComplexOptions(ligName);

foreach (OEMolBase frag in OEBio.OEGetMolComplexComponents(mol, opt))
{
    // ...
```

The process used by the *OESplitMolComplex* and *OEGetMolComplexComponents* functions involves two steps:

1. separating components and assigning functional roles
2. filtering by roles to select the desired components

OESplitMolComplex accesses separate filters for `lig`, `prot`, `wat`, and `other` from the *OESplitMolComplexOptions* object. In *OEGetMolComplexComponents*, the combination of the ligand, protein, and water filters is used to select the output components. The options method *OESplitMolComplexOptions.ResetFilters* can be used for basic changes to the filters, such as selecting a different binding site (see *Listing 11*).

Listing 11: Resetting Filters for All Sites

```
OESplitMolComplexOptions opt = new OESplitMolComplexOptions();
uint allSites = 0;
opt.ResetFilters(allSites);
```

The function *OECountMolComplexSites* returns the number of binding sites.

Listing 12 shows a more involved case, combining protein-complex and binding-site waters into the protein filter and nothing in the water filter.

Listing 12: Modifying Filters

```
OESplitMolComplexOptions opt = new OESplitMolComplexOptions();
OEUnaryRoleSetPred p = opt.GetProteinFilter();
OEUnaryRoleSetPred w = opt.GetWaterFilter();
opt.SetProteinFilter(new OEOrRoleSet(p, w));

opt.SetWaterFilter(
     OEBio.OEMolComplexFilterFactory(OEMolComplexFilterCategory.Nothing));
```

Listing 13 illustrates using an explicit filter when iterating. *OEMolComplexFilter* objects can be of arbitrary complexity.

Listing 13: Explicit Filter When Iterating

```
// for input molecule mol and options opt ...
foreach (OEMolBase l in
               OEBio.OEGetMolComplexComponents(mol,
                                               opt,
                                               opt.GetLigandFilter()))
{
    // ...
```

For more complex workflows, it is useful to explicitly separate the tasks of assigning roles and filtering, if only to avoid the expense of assigning roles multiple times. *Listing 14* shows how to assign roles to each fragment in a molecule for later use.

Listing 14: Assigning Roles to Each Mol Fragment

```
        // for input molecule mol and options opt ...
        OEAtomBondSetVector frags = new OEAtomBondSetVector();

        if (OEBio.OEGetMolComplexFragments(frags, mol, opt))
```

(continues on next page)

(continued from previous page)

```
        {
            //...
```

In *Listing 15*, the role information for each fragment is used to determine the number of binding sites and is filtered to produce ligand and protein complex mols.

Listing 15: Filtering Mol Fragments

```
        // for options opt and OEAtomBondSetVector frags produced earlier ...
        uint numSites = OEBio.OECountMolComplexSites(frags);
        OEChem.OEThrow.Verbose("sites " + numSites);

        OEGraphMol lig = new OEGraphMol();
        OEUnaryRoleSetPred l = opt.GetLigandFilter();
        if (! OEBio.OECombineMolComplexFragments(lig, frags, opt, l))
            OEChem.OEThrow.Warning("Unable to collect ligand from " +
                                   mol.GetTitle());

        OEGraphMol protComplex = new OEGraphMol();
        OEUnaryRoleSetPred p = opt.GetProteinFilter();
        OEUnaryRoleSetPred w = opt.GetWaterFilter();
        if (! OEBio.OECombineMolComplexFragments(protComplex,
                                                 frags,
                                                 opt,
                                                 new OEOrRoleSet(p, w)) )
            OEChem.OEThrow.Warning("Unable to collect complex from " +
                                   mol.GetTitle());
```

The basic idea behind automatic ligand identification is that the list of compounds that are *not* ligands (cofactors, buffers, etc.) grows faster than the list of ligands. The *OEResidueCategoryData* object contained within an *OESplitMolComplexOptions* is a database of these non-ligands. Although this database is being updated as we run across more cofactors, buffers, etc., it is possible for users to update the database as needed. *Listing 16* shows how this is done, involving the Russian nesting doll-like organization of a *OEResidueCategoryData* within a *OEMolComplexCategorizer* within a *OESplitMolComplexOptions* object.

Listing 16: Adding a Cofactor

```
OEResidueCategoryData db = new OEResidueCategoryData();
db.AddToDB(OEResidueDatabaseCategory.Cofactor, "MTQ");

OEMolComplexCategorizer cat = new OEMolComplexCategorizer();
cat.SetResidueCategoryData(db);

OESplitMolComplexOptions opt = new OESplitMolComplexOptions();
opt.SetCategorizer(cat);
```

> **Note**
>
> Please help **OpenEye** maintain and expand the built-in *OEResidueCategoryData* by informing us of any oversights or mistakes.

The most important *OESplitMolComplexOptions* parameters can be controlled from the command line using `OEConfigureSplitMolComplexOptions` and `OESplitMolComplexSetup` (see *Listing 17*).

Listing 17: Command Line Options

```
using System;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class SplitMolComplexDoc
{
    public static int Main(string[] argv)
    {
        OEInterface itf = new OEInterface(interfaceData);
        OEBio.OEConfigureSplitMolComplexOptions(itf);

        if (!OEChem.OEParseCommandLine(itf, argv, "SplitMolComplex"))
            OEChem.OEThrow.Fatal("Unable to interpret command line!");

        OESplitMolComplexOptions opts = new OESplitMolComplexOptions();
        OEBio.OESetupSplitMolComplexOptions(opts, itf);
```

2.3.3 Macromolecule Conformations

Alternate Locations

Because macro-molecular structures are usually represented as static shapes, this gives the mistaken impression that proteins, nucleic acids, etc. are rigid molecules. In truth, these molecules move around quite a lot and in a crystal, loops and other bits are often disordered. Crystallographers work to model multiple conformations in parts of the structure where disorder is observed. If they can, they include a separate copy of each moving atom for each conformation, marking each with an 'alternate location code', a fractional 'occupancy' and a 'temperature factor' quantifying the atom's thermal motion. Section *Biopolymer Residues* discusses how these properties are stored in an *OEResidue*.

Because a structure with alternate locations describes an *ensemble of molecules* rather than a single molecule, they are unsuitable as-is for calculating molecular properties. Often, this is dealt with by dropping all but the first alternate location from the molecule and this is what `OEReadMolecule` does for Protein Data Bank (PDB) files, by default. The first step in dealing with alternate locations is to retain all the alternate location atoms by setting the input flavor before reading the molecule, as shown below.

```
ims.SetFlavor(OEFormat.PDB, OEIFlavor.PDB.ALTLOC);
```

With all the alternate atoms retained, you can use the predicate *OEHasAlternateLocation* to identify these atoms. Although alternate locations are atom properties, they usually describe the coordinated motion of groups of atoms. Each connected set of atoms with alternate location codes that move in a coordinated fashion is called an *alternate location group* (represented by an `OEAltGroup`) and each conformation of a group's atoms is called an *alternate location* (represented by an `OEAltLocation`). The second step in dealing with alternate locations is to use the *OEAltLocationFactory*, a class that will manage these groups and locations for you.

Listing 1: Alternate location factory groups

```
using System;
using OpenEye.OEBio;
using OpenEye.OEChem;

public class AltlocfactGroupsDoc
{
    public static void PrintAltGroupInfo(OEMolBase mol)
    {
        if (! OEChem.OEHasResidues(mol))
            OEChem.OEPerceiveResidues(mol, OEPreserveResInfo.All);

        // create factory for mol
        OEAltLocationFactory alf = new OEAltLocationFactory(mol);

        Console.WriteLine(mol.GetTitle() + "\t"
                          + "(" + alf.GetGroupCount() + " groups)");

        foreach (OEAltGroup grp in alf.GetGroups())
        {
            Console.WriteLine("\t" + grp.GetLocationCount() + " locs"
                              + ":" + alf.GetLocationCodes(grp));
        }
    }

    public static int Main(string[] args)
    {
        if (args.Length != 1)
        {
            OEChem.OEThrow.Usage("AltlocfactGroupsDoc <mol-infile>");
        }

        oemolistream ims = new oemolistream();
        if (!ims.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        // need this flavor to read alt loc atoms
        ims.SetFlavor(OEFormat.PDB, OEIFlavor.PDB.ALTLOC);

        foreach (OEGraphMol mol in ims.GetOEGraphMols())
        {
            PrintAltGroupInfo(mol);
        }
        return 0;
    }
}
```

In addition to providing methods to work with alternate locations and groups, the *OEAltLocationFactory* corrects its copy of the input source molecule for bond and formal charge problems caused by atoms having multiple locations (something the standard molecule perception routines are not setup to handle).

The *OEAltLocationFactory* also provides methods for manufacturing subset molecules that represent specific selec-

tions among the different groups of alternate locations. The initial (*primary*) selection is the alternate location in each group with the largest average occupancy. In the example below, the subset is for the previous set of location selections plus the location with code `'B'` that includes the residue of the specified atom.

Listing 2: Making an alternate location factory subset mol

```
// given OEAltLocationFactory alf and OEAtomBaseIter atom ...
OEAltLocation loc = alf.GetLocation(atom.Target(), 'B');
OEGraphMol ssmol = new OEGraphMol();
if (alf.MakeAltMol(ssmol, loc))
{
    // use the subset mol...
```

Dihedrals and Sidechain Rotamers

The function *OEGetRotamers* returns an iterator of 'rotameric' sidechain conformations for a given amino-acid type, while *OESetRotamer* will set the sidechain chi angles of a specified *OEHierResidue* or *OEAtomBase* to a given rotamer. The *Dunbrack* [Dunbrack-1997], *Richardson* [Lovell-2000], and newer *Richardson_2016* [Hintze-2016] rotamer libraries are supported.

There are also functions that return backbone and sidechain dihedral angles (*OEGetPhi*, *OEGetPsi*, *OEGetChis*, *OEGetTorsion*) and modify dihedrals (*OESetTorsion*).

Swapping Ambiguous Isoelectronic Residue Atoms

The function *OESwapAIEResidueAtoms* exchanges the coordinates of nitrogen and oxygen atoms in aspartic acid, asparagine, glutamic acid and glutamine sidechains and the ND1/CD2 and CE1/NE2 atoms in histidine rings. These are atoms that may be confused with one another in an electron density map because they have the same or very similar electron density and swapping coordinates is occasionally required to correct an error in a structure.

2.3.4 Interaction Perception

OEBio TK provides the following APIs to perceive protein-ligand interactions:

OEPerceiveInteractionHints
: The function that perceives protein-ligand interactions.

OEPerceiveInteractionOptions
: The class that stores the parameters that control the interaction perception.

  The default geometric parameters have been set based on literature data ([Kumar-2002], [Cavallo-2016], [Bissantz-2010], and [Marcou-2007]).

Interaction Hint Container

Interactions perceived by the *OEPerceiveInteractionHints* function are stored in an *OEInteractionHintContainer* object. The architecture of the *OEInteractionHintContainer* class is depicted in *Schematic representation of interaction hint container*.

Other related APIs:

OEIsValidActiveSite
: This function checks whether the interaction container initialized correctly as **active site** i.e. containing two molecules: one typed as *ligand* and one typed as *protein*.

OEGetActiveSiteInteractionHintTypes
: This function iterates over all interaction types (including subtypes) that are perceived by the *OEPerceiveInteractionHints* function.

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Depicting Active Site Interactions **Grapheme TK** example

OpenEye Python Cookbook recipes:

- Accessing Interaction Hint Information
- Print Protein-Ligand Interactions to Console
- Print the Summary of Protein-Ligand Interactions for a Set of Molecules

Interaction Hint Predicates

The table below list all interaction predicates that are currently available in **OEBio TK**.

Table 21: **OEBio TK interaction predicates**

Access interactions	Functor Name
with given interaction type	*OEHasInteractionHintType*
with given atom	*OEHasInteractionHint*
with given residue	*OEHasResidueInteractionHint*
contact	*OEIsContactInteractionHint*
hydrogen bond	*OEIsHBondInteractionHint*
inter-molecular hydrogen bond	*OEIsIntermolecularHBondInteractionHint*
intra-molecular hydrogen bond	*OEIsIntramolecularHBondInteractionHint*
chelator	*OEIsChelatorInteractionHint*
inter-molecular chelator	*OEIsIntermolecularChelatorInteractionHint*
intra-molecular chelator	*OEIsIntramolecularChelatorInteractionHint*
covalent	*OEIsCovalentInteractionHint*
clash	*OEIsClashInteractionHint*
clashing hydrogen bond	*OEIsClashingHBondInteractionHint*
salt-bridge	*OEIsSaltBridgeInteractionHint*
inter-molecular salt-bridge	*OEIsIntermolecularSaltBridgeInteractionHint*
intra-molecular salt-bridge	*OEIsIntramolecularSaltBridgeInteractionHint*
T and Pi-stacking	*OEIsStackingInteractionHint*
inter-molecular stacking	*OEIsIntermolecularStackingInteractionHint*
intra-molecular stacking	*OEIsIntramolecularStackingInteractionHint*
cation-Pi	*OEIsCationPiInteractionHint*
inter-molecular cation-Pi	*OEIsIntermolecularCationPiInteractionHint*
intra-molecular cation-Pi	*OEIsIntramolecularCationPiInteractionHint*
halogen bond	*OEIsHalogenBondInteractionHint*
inter-molecular halogen bond	*OEIsIntermolecularHalogenBondInteractionHint*
intra-molecular halogen bond	*OEIsIntramolecularHalogenBondInteractionHint*
inter-molecular	*OEIsInterInteractionHint*
intra-molecular	*OEIsIntraInteractionHint*
non-ideal hydrogen bond	*OEIsNonIdealHBondInteractionHint*
unpaired (hbond or salt-bridge)	*OEIsUnpairedInteractionHint*
unpaired ligand	*OEIsUnpairedLigandInteractionHint*
unpaired protein	*OEIsUnpairedProteinInteractionHint*

The following logical operations allow to combine both the built-in and the user defined interaction predicates.

Table 22: **Interaction Composite Functors**

Composite Functor	Description
OENotInteractionHint	logical not
OEOrInteractionHint	logical or
OEAndInteractionHint	logical and

Accessing Calculated Interaction Hint Geometries

When interaction hints are perceived using the *OEPerceiveInteractionHints* function, various geometries (such as distances and angles) are calculated that are checked against the geometric constraints stored in the given *OEPerceiveInteractionOptions* class.

The following code snippet shows how to access these calculated geometries:

```
OEBio.OEPerceiveInteractionHints(asite);
OEHBondInteractionHint hbond = new OEHBondInteractionHint(OEHBondInteractionHintType.
↪LigandDonates);
OEHasInteractionHintType hbondpred = new OEHasInteractionHintType(hbond);

foreach (OEInteractionHint ih in asite.GetInteractions(hbondpred))
{
    OEInteractionHintTypeBase itype = ih.GetInteractionType();
    Console.WriteLine("Calculated geometries for '" + itype.GetName() + "' interaction␣
↪type:");
    foreach (String gname in ih.GetCalculatedGeometries())
        Console.WriteLine(gname + ":\t" +
            String.Format("{0,4:0.000}", ih.GetCalculatedGeometry(gname)));
}
```

For the *OEHBondInteractionHintType.LigandDonates* hydrogen bond interaction type, three geometries are calculated:

- the distance between the acceptor and the donor heavy atoms (see also the *OEPerceiveInteractionOptions.SetMaxHBondDistance* method)
- the acceptor angle (see also the *OEPerceiveInteractionOptions.SetMaxAcceptorAngle* method)
- the donor angle (see also the *OEPerceiveInteractionOptions.SetMaxDonorAngle* method)

The "distance" values are always given in Angstroms and the "angle" values are always given in radians. The output of the code snippet above might look like this:

```
Calculated geometries for 'bio:active-site:hbond:ligand2protein' interaction type:
bio:interaction:distance       : 2.722
bio:intaraction:acceptor-angle : 0.155
bio:interaction:donor-angle    : 0.241
```

For different interactions, different geometries are calculated. The following snippet shows how to access the names of the geometries that are available for a given interaction type using the *OEGetCalculatedInteractionGeometries* function:

```
OEHBondInteractionHint hbondacc = new OEHBondInteractionHint(OEHBondInteractionHintType.
↪LigandAccepts);
Console.WriteLine("Calculated geometries for '" + hbondacc.GetName() + "' interaction␣
```

(continues on next page)

(continued from previous page)

```
↪type:");
foreach (String geom in OEBio.OEGetCalculatedInteractionGeometries(hbondacc))
    Console.WriteLine(geom);
```

All interaction geometry names start with the `bio:interaction:` prefix. However, both the *OEInteractionHint.HasCalculatedGeometry* and *OEInteractionHint.GetCalculatedGeometry* methods accept names with or without the `bio:interaction:` prefix.

The following code snippet shows how to access the calculated geometries directly:

```
if (inter.HasCalculatedGeometry("distance"))
{
    double distance = inter.GetCalculatedGeometry("distance");
    Console.WriteLine("distance: " +
                      String.Format("{0,4:0.000}", distance) + " Angstrom(s)");
}
if (inter.HasCalculatedGeometry("acceptor-angle"))
{
    double angle = ToDegrees(inter.GetCalculatedGeometry("acceptor-angle"));
    Console.WriteLine("acceptor angle: " +
                      String.Format("{0,4:0.000}", angle) + " degree(s)");
}
```

The output of the code snippet above might look like this:

```
distance: 2.722  Angstrom(s)
acceptor angle: 8.863 degree(s)
```

Hint

It is a good practice to call *OEInteractionHint.HasCalculatedGeometry* prior to invoking the *OEInteractionHint.GetCalculatedGeometry* method to check whether the interaction hint has a geometry with a given name.

Currently, the following names are used for the interaction geometries:

- `bio:interaction:distance`
- `bio:interaction:angle`
- `bio:interaction:acceptor-angle`
- `bio:interaction:donor-angle`

For most interactions (except the unpaired ones), the calculated distance is available by using the `bio:interaction:distance` name. None of the unpaired interactions have any calculated geometries since they only "suggest" how they could contribute to the binding but are not actually formed in the complex.

A calculated geometry, such as a distance, can have different meanings based on the interaction type. For example, in a hydrogen bond interaction it means the distance between the acceptor and donor heavy atoms. For a pi-staking interaction, it means the distance between the center of the two ring systems. The following table shows the correspondence between the calculated geometries and the geometric constraints.

Set method	Interaction type	Associated geometry name
SetMaxHBondDistance	hbond	`bio:interaction:distance`
SetMaxChargeAidedHBondDistanc	hbond (charged)	`bio:interaction:distance`
SetMaxNonIdealHBondDistance	non-ideal hbond	`bio:interaction:distance`
SetMaxAcceptorAngle	hbond	`bio:interaction:acceptor-angle`
SetMaxDonorAngle	hbond	`bio:interaction:donor-angle`
SetMaxNonIdealAcceptorAngle	non-ideal hbond	`bio:interaction:acceptor-angle`
SetMaxNonIdealDonorAngle	non-ideal hbond	`bio:interaction:donor-angle`
SetMaxMetalBondDistance	chelator	`bio:interaction:distance`
SetMaxSaltBridgeDistance	salt bridge	`bio:interaction:distance`
SetMaxCationPiDistance	cation-Pi	`bio:interaction:distance`
SetMaxCationPiAngle	cation-Pi	`bio:interaction:angle`
SetMaxPiStackDistance	Pi-stacking	`bio:interaction:distance`
SetMaxPiStackAngle	Pi-stacking	`bio:interaction:angle`
SetMaxTStackDistance	T-stacking	`bio:interaction:distance`
SetMaxTStackAngle	T-stacking	`bio:interaction:angle`
SetMinStackOffset	Pi-stacking and T-stacking	(no geometry — acts as a filter)
SetMaxHalogenBondDistance	halogen bond	`bio:interaction:distance`
SetMaxHalogenAngle	halogen bond	`bio:interaction:angle`

2.3.5 Receptors

In receptor objects contain information about the location and characteristics of the binding pocket, and required for docking (see Docking), posit (see POSIT), and scoring (see Scoring).

A receptor is an *OEReceptor* object that resides within a *OEDesignUnit*. The essential component of a valid receptor is the negative image that describes the shape of the active site.

Creating a Receptor

Within the Docking toolkit the `OEMakeReceptor` function is used to create a receptor from the structure of a target protein containing in a *OEDesignUnit*.

The protein structure should only include molecules the ligand is expected to interact with. In general crystallographic waters, other solvents and the bound ligand should be not be included as part of the target protein structure passed to the `OEMakeReceptor` function, although in certain cases, the user may wish to retain certain key molecules as part of the target protein structure (*e.g.* a crystallographic water).

Contents of a Receptor

Negative Image

The negative image describes the shape of the active site. It is stored as a potential grid surrounding the active site. Potential values are always greater than or equal to zero. The negative image has high potentials where ligand atoms make many contacts with atoms of the active site without clashing and in positions some ligand atoms are likely to occupy when other atoms of the ligand make good contacts with the receptor (*e.g.* bridging positions ligand atoms will likely occupy when a ligand is stretched between two pockets).

During docking two shapes are created by contouring the negative image potential grid. The two shapes control the docking process as follows:

Outer Contour Shape

> During docking any pose examined by the exhaustive search that does not fit within this shape will be rejected. A pose is considered to fit if the center of every heavy atom is within this shape. The volume of this shape is typically between 500 and 2000 cubic Angstroms.

Inner Contour Shape

> During docking any pose examined by the exhaustive search that does not touch this shape is rejected. A pose is considered to touch if the center of at least one heavy atom falls within this shape. The volume of this shape is typically 50 to 100 cubic angstroms.

While neither the *outer contour* nor the *inner contour* shape are required, it is recommended that the *outer contour* always be used (docking speed will be dramatically slower without the *outer contour*). Using the *inner contour* improves docking speed slightly at the expense of sampling. However, if the *inner contour* size is nearly as large as the *outer contour*, there will be little to no difference in compute time.

The negative image of the receptor is set up when the receptor is created with the `OEMakeReceptor` function (see the section on creating a receptor). The size of the *outer contour* and *inner contour* shapes are controllable by setting the contour level used to create the shape from the negative image. The `OEMakeReceptor` function will automatically set a reasonable contour level for both the *inner contour* and *outer contour*, however the *inner contour* will be disabled by setting the value to be negative (see below).

The negative image grid has only positive values, thus only contour levels that are positive create a shape. When either the *outer contour* level or *inner contour* level are negative that contour shape will be disabled (*i.e.* ignored during the docking process).

The volume of the *outer contour* shape has a significant effect on docking speed, while the volume of the *inner contour* shape has a modest effect on docking speed. In both cases the smaller volumes increase docking speed by reducing the number of poses that are scored. In general an outer and inner contour volume of between 500-2000 and 50-100 cubic Angstroms respectively is recommended.

Constraints

Constraints are key interactions ligands are required to make when docking into the active site. They are optional and user defined.

Constraints do not affect how a given pose scores, however they do affect how the docking algorithm chooses poses to score during the docking process. Any pose that does not match the docking constraints will be rejected and replaced by the next best scoring pose. If no poses of a ligand match the docking constraints the ligand will not be docked. If multiple constraints are specified every constraint must be satisfied or the pose will be rejected.

> **Note**
>
> The docking process has a resolution of approximately 1 Ångström, and thus the constraints have a similar resolution. Therefore it is possible that poses docked with a constraint may have small violations of the constraint distance, up to approximately 1 Ångström.

Receptors support two general classes of constraints; protein constraints and custom constraints. There may be any number of either class of constraint.

> **Note**
>
> Each individual custom or protein constraint has an enabled flag. A disabled constraint is ignored during the docking process.

Protein Constraints

A protein constraint specifies an interaction that must be made with a **heavy atom** of the protein structure (*i.e.* protein constraints cannot be placed on hydrogen atoms). There are five types of protein constraints.

Contact

A contact constraint is satisfied when any ligand heavy atom is within 4 angstroms of the protein heavy atom.

Lipophilic

A lipophilic constraint is satisfied when any non-polar heavy atom on the ligand is within 4 angstroms of the protein heavy atom.

Donor

A donor constraint is satisfied when a *donor on the ligand* makes a hydrogen bond interaction with the protein heavy atom.

Acceptor

An acceptor constraint is satisfied when an *acceptor on the ligand* makes a hydrogen bond interaction with the protein heavy atom. Acceptor constraints must be placed on the protein heavy atom the donor hydrogen is interacting with.

Chelator

A chelator constraint is satisfied when a *chelator on the ligand* makes a metal-chelator interaction with the protein heavy atom.

Only one protein constraint is allowed per protein heavy atom. If a protein constraint is set on a protein atom that already has a protein constraint the original protein constraint will be discarded and replaced by the new constraint.

Custom Constraints

A custom constraint consists of one or more spheres within the receptor active site, and optionally a list of SMARTS patterns. A custom constraint is satisfied when a matching atom on the ligand falls within any of the spheres associated with the custom constraint. If no SMARTS patterns are specified any heavy atom on the ligand can satisfy the constraint, otherwise only atoms that match one of the smarts patterns can satisfy the constraint.

2.4 OEGrid Theory

2.4.1 Scalar Grids

A grid is a data container that holds a value at every point in a three dimensional lattice. The simplest type of value to hold is a floating point value. In **OEChem TK**, these are called `OEScalarGrids`. `OEScalarGrids` are inherently cubic.

The following fundamental parameters define a grid:

Dimension
The number of grid points per axis

Midpoint
The center of the grid in Cartesian space

Spacing
The length between grid points

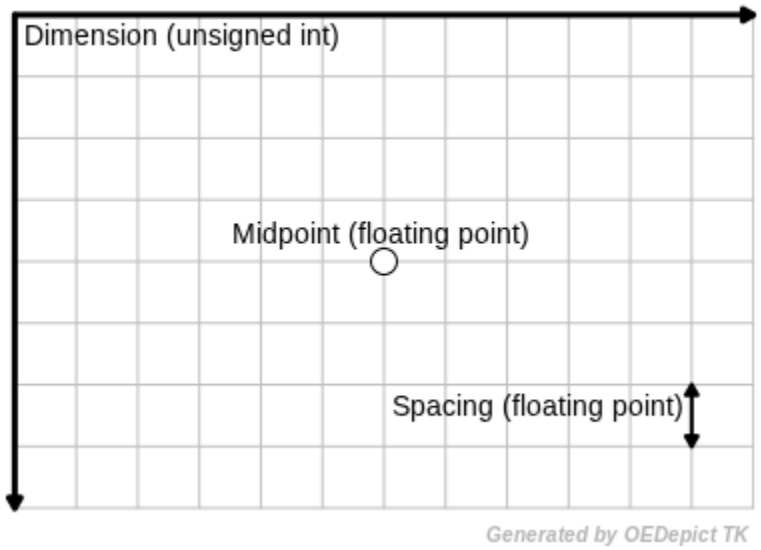


Fig. 80: **Fundamental Grid Parameters**

Listing 1 demonstrates how to *construct* a grid with 64 points in each direction, a midpoint at the origin, and a spacing of 0.5 between each point in the grid.

Listing 1: Constructing a scalar grid

```
using System;
using OpenEye.OEChem;
using OpenEye.OEGrid;

public class ScalarGridCtor
{
    public static int Main(string[] args)
    {
        OEScalarGrid grid = new OEScalarGrid(64, 64, 64, 0.0f, 0.0f, 0.0f, 0.5f);
        return 0;
    }
}
```

Note

The memory footprint of the grid is determined by the dimensions. It is easy to create grids that are too large for memory. For example, the grid in *Listing 1* will consume $64^3 * sizeof(float) = 1048576bytes = 1MB$ of memory for this nominally sized grid.

Minimum and maximum

Every grid can have its maximum and minimum vales reported by GetXMax, GetXMin etc. The image `Grid coordinates` shows a 2D representation for simplicity. It shows 4 grid points centered at (0,0), with a spacing of 1.0. Each grid point will have spatial coordinates (0.5,0.5), (-0.5,0.5) etc. However each grid point lies at the center of a pixel (or rather a voxel in 3D). Any property that falls into the area (volume) of that grid point is mapped to that grid point. This means that the maximum value of x is the extent of the grid, not just the maximum value of the grid point coordinates. In this case that value is 1.0.

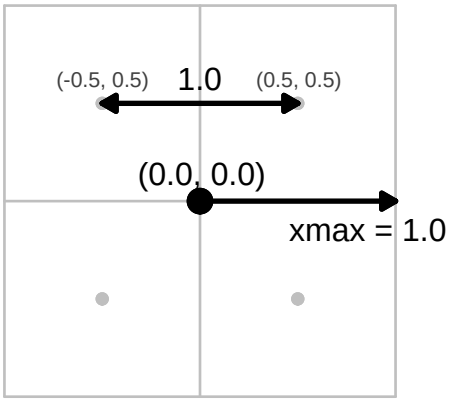


Fig. 81: **Grid coordinates**

Data Access

Grids provide three ways to index data inside the grid. They are ordered here by speed of access, i.e., grid elements perform simple pointer arithmetic where spatial coordinates require floating point computation.

Grid Element
: Index into the underlying data array

Grid Indices
: Unsigned integers less than the dimension for that axis

Spatial Coordinates
: Cartesian coordinates of any point inside the grid, the grid will determine the closest grid point

Spatial Coordinates

Assuming `grid` is already filled with relevant values, *Listing 2* demonstrates how to retrieve the grid value closest to the atoms in the molecule `mol`.

Listing 2: Getting values associated with coordinates

```
foreach (OEAtomBase atom in mol.GetAtoms())
{
    float[] xyz = new float[3];
    mol.GetCoords(atom, xyz);
```

(continues on next page)

(continued from previous page)

```
    if (grid.IsInGrid(xyz[0], xyz[1], xyz[2]))
    {
        Console.WriteLine("value = {0:0.00000}",
                          grid.GetValue(xyz[0], xyz[1], xyz[2]));
    }
}
```

Spatial coordinates can also be used to set data on the grid. *Listing 3* demonstrates how to assign an atom's partial charge to the grid point it is closest to.

Listing 3: Setting values associated with coordinates

```
foreach (OEAtomBase atom in mol.GetAtoms())
{
    float[] xyz = new float[3];
    mol.GetCoords(atom, xyz);
    if (grid.IsInGrid(xyz[0], xyz[1], xyz[2]))
    {
        float charge = (float)atom.GetPartialCharge();
        grid.SetValue(xyz[0], xyz[1], xyz[2], charge);
    }
}
```

Note

In the preceding two code fragments bounds checking was explicitly performed using the *IsInGrid* method. Accessing data outside the grid is undefined behavior (usually a segmentation fault). However, *IsInGrid* can become an expensive operation if performed excessively. One way to avoid this cost is to make sure your grid is big enough to enclose the object being worked on. See *OEMakeGridFromCenterAndExtents* for an example of constructing a grid that covers the entire molecule to ensure no spatial coordinate access is outside the bounds of the grid.

Grid Indices

Grid indices are faster than spatial coordinates because there is no floating point arithmetic to perform. Grid indices make it easy to iterate over the neighbors of any particular point in the grid. *Listing 4* demonstrates iterating over all 27 grid points adjacent to and including the grid point given by the grid indices: `ix`, `iy`, `iz`.

Listing 4: Iterating over neighbor points

```
float[] xyz = new float[3];
mol.GetCoords(atom, xyz);
uint[] ixyz = new uint[3];
grid.SpatialCoordToGridIdx(xyz[0], xyz[1], xyz[2], ixyz);

// Make sure not to go past grid bounds
uint mini = Math.Max(ixyz[0] - 1, 0);
uint minj = Math.Max(ixyz[1] - 1, 0);
uint mink = Math.Max(ixyz[2] - 1, 0);
uint maxi = Math.Min(ixyz[0] + 1, grid.GetXDim());
uint maxj = Math.Min(ixyz[1] + 1, grid.GetYDim());
uint maxk = Math.Min(ixyz[2] + 1, grid.GetZDim());
```

(continues on next page)

(continued from previous page)

```
for (uint k = mink; k < maxk; k++)
{
    for (uint j = minj; j < maxj; j++)
    {
        for (uint i = mini; i < maxi; i++)
        {
            Console.WriteLine("value = {0:0.00000}", grid.GetValue(i, j, k));
        }
    }
}
```

Grid Elements

Grid values are actually stored in a large one dimensional block of memory. The fastest way to access all the data is to linearly scan through memory. *Listing 5* demonstrates how to square every value in the grid.

Listing 5: Squaring every grid value

```
for (uint i = 0; i < grid.GetSize(); ++i)
{
    float val = grid.GetValue(i);
    grid.SetValue(i, val * val);
}
```

Input and Output

The following grid file formats are supported:

Table 23: **OEChem TK grid file format support**

Grid File Type (or reader)	File Extension(s)	Description	Read	Write
OEGridFileType.Grasp	.phi	GRASP	Yes	Yes
OEGridFileType.GRD	.grd	OpenEye Binary format	Yes	Yes
OEGridFileType.Ascii	.agd	OpenEye ASCII format	Yes	Yes
OEGridFileType.CCP4	.map .ccp .ccp4	CCP4 format	Yes	Yes
OEGridFileType.XPLOR	.xplor .xplmap	XPLOR format	Yes	No
OEReadMTZ	.mtz	MTZ crystallographic reflection format	Yes	No

The ASCII format (`.agd`) was developed by OpenEye to allow for easy integration with other software. The following is an example of the ASCII output for a grid centered at the origin, 2 points along each axis, a spacing of 0.5, and every value zero.

```
Title: Example Grid
Mid:     0.000000     0.000000     0.000000
Dim:      2      2      2
Spacing:     0.500000
Values:
0.000000e+00
0.000000e+00
0.000000e+00
```

(continues on next page)

(continued from previous page)

```
0.000000e+00
0.000000e+00
0.000000e+00
0.000000e+00
0.000000e+00
```

Listing 6 is the code used to write the ASCII format out. It is provided here to leave no doubt as to how to interoperate with the format.

Listing 6: Writing the ASCII format

```
using System;
using OpenEye.OEChem;
using OpenEye.OEGrid;

public class ASCIIWriter
{
    public static int Main(string[] args)
    {
        OEScalarGrid grid = new OEScalarGrid(2, 2, 2, 0.0f, 0.0f, 0.0f, 0.5f);
        grid.SetTitle("Simple Grid");

        Console.WriteLine(String.Format("Title: {0}", grid.GetTitle()));

        Console.WriteLine("Mid: {0,12:0.000000} {1,12:0.000000} {2,12:0.000000}",
                          grid.GetXMid(),
                          grid.GetYMid(),
                          grid.GetZMid());

        Console.WriteLine("Dim: {0,6:d} {0,6:d} {0,6:d}",
                          grid.GetXDim(),
                          grid.GetYDim(),
                          grid.GetZDim());

        Console.WriteLine("Spacing: {0,12:0.000000}", grid.GetSpacing());

        Console.WriteLine("Values:");
        for (uint iz = 0; iz < grid.GetZDim(); iz++)
        {
            for (uint iy = 0; iy < grid.GetYDim(); iy++)
            {
                for (uint ix = 0; ix < grid.GetXDim(); ix++)
                {
                    Console.WriteLine("{0:e6}", grid.GetValue(ix, iy, iz));
                }
            }
        }
        return 0;
    }
}
```

Grid as Generic Data

Grids can also be attached to molecules and then written out to OEBinary (`.oeb`) files. A visualizer can then read in the molecule and grid without any other means of making the association. *Listing 7* demonstrates how to attach a Gaussian grid to the molecule it was created from.

The grid is attached to a molecule using the 'generic data' interface provided by the *OEBase* base class. This allows an arbitrary number of grids (or any type of data) to be attached to molecules. Since all grids also derive from *OEBase* generic data can be attached to grids as well allowing for arbitrarily complex data hierarchies.

Listing 7: Attaching a grid to a molecule

```
OEScalarGrid grid = new OEScalarGrid();
OEGrid.OEMakeMolecularGaussianGrid(grid, mol, 0.5f);

OEGrid.OESetGrid(mol, "Gaussian Grid", grid);

OEChem.OEWriteMolecule(ofs, mol);
```

See also

- *Generic Data* chapter

CHAPTER

THREE

EXAMPLES

3.1 OEChem Examples

The following table lists the currently available **OEChem TK** examples.

3.1.1 Molecule Processing

Program	Description
Convert	converting molecule files
CatMols	concatenating molecules
Parts2Mols	splitting multicomponent molecules
MolExtract	extracting molecules by title
UniqMol	writing out unique molecules (canonical SMILES)
UniqInchi	writing out unique molecules (InChI)
RandomizeAtoms	randomizing atoms of molecules
CanSmi	generating canonical smiles
SizeFilter	filtering molecules by weight or heavy atom count
StripSalts	stripping salt
RingSubset	extracting rings
ExtractScaffold	extracting molecule scaffolds
RandomSample	extracting random molecule subset
MolChunk	split molecule file into N chunks or chunks of size N

3.1.2 Reactions

Program	Description
UniMolRxn	performing a reaction
LibGen	generating a library

3.1.3 Molecule Searching

Program	Description
MolGrep	performing substructure search

3.1.4 Molecule Alignment

Program	Description
MCS3DAlign	aligning molecules by maximum common substructure
CliqueAlign	aligning molecules by clique match
SMARTSAlign	aligning molecules by SMARTS match
RMSD	aligning multiconformer molecules

3.1.5 SDF Specific

Program	Description
SDFModProps	modifying SD tags
SDF2CSV	exporting SD data to a CSV file
CSV2SDF	adding CSV data as SD tags
sdfrename	renaming molecules by SD field
sdfilter	filtering molecules by SD data

3.1.6 Molecule Information Examples

Program	Description
MolCount	counting molecules
GetTitles	get molecule titles
MolStats	printing simple molecule information
RotorCount	printing rotor count distribution
MinPath	finding minimum path in a molecule
PrintAtomNames	printing molecule atom names

3.1.7 2D Coordinate Generation Examples

Program	Description
ExtractRingTemplates	extracting 2D ring templates
CreateRingDict	creating 2D ring dictionary
AppendRingDict	appending 2D ring dictionary
Generate2D	generating 2D coordinates

3.1.8 3D Molecule Styling Examples

3.1.9 Miscellaneous Examples

Program	Description
OEChemInfo	printing OEChem and format information
Anisou	printing anisotropic B-factors
Dots	printing dots
Options	defining parameters

Examples

Convert molecule files

A program that converts molecules from one format to another based on the file extension.

Example

```
prompt> Convert input.mol2 output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Program to convert from one molecule format to another
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class Convert
{
    public static void Main(string[] args)
    {
        if (args.Length != 2)
        {
            OEChem.OEThrow.Usage("Convert <infile> <outfile>");
        }
        ConvertFile(args[0], args[1]);
    }

    private static void ConvertFile(string infile, string outfile)
    {
        OEMol mol = new OEMol();

        oemolistream ifs = new oemolistream();
        if (!ifs.open(infile))
        {
            OEChem.OEThrow.Fatal("Unable to open " + infile + " for reading");
```

(continues on next page)

(continued from previous page)

```
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(outfile))
        {
            OEChem.OEThrow.Fatal("Unable to open " + outfile + " for writing");
        }

        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

See also

- *OEReadMolecule* function
- *OEWriteMolecule* function

Concatenating molecules

A program that takes an input molecule file (or files) and concatenates all the molecules into a single molecule file.

Example

```
prompt> CatMols -i file1.oeb.gz file2.oeb.gz -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
```

(continues on next page)

(continued from previous page)

```
* This program concatenates molecules into one file.
* It can be useful for generating ROCS queries or reattach ligands to an
* protein structure
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class CatMols
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "CatMols", args);

        OEGraphMol omol = new OEGraphMol();
        foreach (string filename in itf.GetStringList("-i"))
        {
            oemolistream ifs = new oemolistream();
            if (ifs.open(filename))
            {
                OEGraphMol imol = new OEGraphMol();
                while (OEChem.OEReadMolecule(ifs, imol))
                {
                    OEChem.OEAddMols(omol, imol);
                }
            }
            else
            {
                OEChem.OEThrow.Fatal("Unable to open " + filename + " for reading");
            }
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(itf.GetString("-o")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for writing
↪");
        }

        OEChem.OEWriteMolecule(ofs, omol);
    }

    private static string interfaceData=@"
!BRIEF -i <infile1> [<infile2>...] -o <outfile>
!PARAMETER -i
  !ALIAS -in
  !TYPE string
  !LIST true
  !REQUIRED true
  !BRIEF input file name(s)
!END
!PARAMETER -o
  !ALIAS -out
```

(continues on next page)

(continued from previous page)

```
  !TYPE string
  !REQUIRED true
  !BRIEF output file name
!END
";
}
```

See also

- *OEAddMols* function
- *OEInterface.GetList* method

Splitting multicomponent molecules

A program that splits multicomponent molecules into their constituent parts. The program will output the molecules to stdout as SMILES if no output file is set. The program will print the number of input and output molecules to stderr.

Example

```
prompt> Parts2Mols inputfile1.oeb.gz output.oeb.gz
```

If the output-file argument is omitted, the output is written to standard output.

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Writes each component of a molecule as a separate molecule
//****************************************************************************/
using System;
using OpenEye.OEChem;

public class Parts2Mols
```

(continues on next page)

(continued from previous page)

```
{
    public static void Main(string[] args)
    {
        if (args.Length < 1 || args.Length > 2)
        {
            OEChem.OEThrow.Usage("Parts2Mols <infile> [<outfile>]");
        }

        oemolistream ifs = new oemolistream();
        ifs.SetFlavor(OEFormat.PDB, OEIFlavor.PDB.Default ^ OEIFlavor.PDB.DATA);
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        oemolostream ofs = new oemolostream(".ism");
        if (args.Length == 2)
        {
            if (!ofs.open(args[1]))
            {
                OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
            }
        }

        int incount = 0;
        int outcount = 0;
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            incount++;
            uint[] parts = new uint[mol.GetMaxAtomIdx()];
            uint pcount = OEChem.OEDetermineComponents(mol, parts);
            OEPartPredAtom pred = new OEPartPredAtom(parts, (uint)parts.Length);
            for (uint i = 1; i <= pcount; ++i)
            {
                outcount++;
                pred.SelectPart(i);
                OEGraphMol partmol = new OEGraphMol();
                bool adjustHCount = true;
                OEChem.OESubsetMol(partmol, mol, pred, adjustHCount);
                OEChem.OEWriteMolecule(ofs, partmol);
            }
        }
        Console.Error.WriteLine("results:  in: " + incount + "  out: " + outcount);
    }
}
```

See also

- *Connected Components Identification* section
- *OEDetermineComponents* function

- *OESubsetMol* function

Extract molecules by title

A program that extracts molecules from a file based on their title. The program can take as input either a specific title or a file containing a list of titles.

Example

```
prompt> MolExtract -title Mol0001 -i dbase.oeb.gz -o Mol0001.oeb.gz
prompt> MolExtract -list names.txt dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Extract compound(s) from a file based on molecule title
//****************************************************************************
using System;
using System.IO;
using System.Collections.Generic;
using OpenEye.OEChem;

public class MolExtract
{
    private static HashSet<string> FillTitleSet(OEInterface itf)
    {
        HashSet<string> titles = new HashSet<string>();

        if (itf.HasString("-title"))
        {
            titles.Add(itf.GetString("-title"));
        }
        else if (itf.HasString("-list"))
```

(continues on next page)

(continued from previous page)

```
        {
            StreamReader sr = null;
            try
            {
                sr = new StreamReader(itf.GetString("-list"));
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    titles.Add(line.Trim());
                }
            }
            catch
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-list") + " for␣
↪reading");
            }
            finally
            {
                if (sr != null)
                {
                    sr.Close();
                }

            }
        }
        else
        {
            OEChem.OEThrow.Usage("Must give either -list or -title");
        }
        return titles;
    }

    private static void Extract(OEInterface itf)
    {
        HashSet<string> titleSet = FillTitleSet(itf);
        if (titleSet.Count == 0)
        {
            OEChem.OEThrow.Fatal("No titles requested");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(itf.GetString("-o")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for writing
↪");
```

(continues on next page)

(continued from previous page)

```
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (titleSet.Contains(mol.GetTitle()))
            {
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }
    }

    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "MolExtract", args);
        Extract(itf);
    }

    private static string interfaceData=@"
!BRIEF -title title | -list <moltitles.file> [-i] <input> [-o] <output>
!PARAMETER -i
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !ALIAS -out
  !TYPE string
  !REQUIRED true
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -title
  !ALIAS -t
  !TYPE string
  !BRIEF Single mol title to extract
!END
!PARAMETER -list
  !ALIAS -l
  !TYPE string
  !BRIEF List file of mol titles to extract
!END
";
}
```

See also

- *OEMolBase.GetTitle* method

Write out unique molecules (canonical SMILES)

A program that loads a database of molecules and outputs those that are unique. Uniqueness is defined by whether they have the same canonical isomeric SMILES.

Example

```
prompt> UniqMol dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Read molecules and write out the unique ones. Two molecules are considered
// identical if their canonical isomeric smiles are identical.
//****************************************************************************/
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class UniqMol
{
    public static void Main(string[] args)
    {
        if (args.Length != 2)
        {
            OEChem.OEThrow.Usage("UniqMol <infile> <outfile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
```

(continues on next page)

(continued from previous page)

```
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
        }
        ProcessFile(ifs, ofs);
    }

    private static void ProcessFile(oemolistream ifs, oemolostream ofs)
    {
        OEGraphMol mol = new OEGraphMol();
        HashSet<string> UniqSmiles = new HashSet<string>();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            string smi = OEChem.OECreateIsoSmiString(mol);
            if (UniqSmiles.Add(smi))
            {
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }
    }
}
```

See also

- *OECreateIsoSmiString* function
- *Write out unique molecules (InChI)* section

Write out unique molecules (InChI)

A program that loads a database of molecules and outputs those that are unique. Uniqueness is defined by whether they have the same standard InChI.

Example

```
prompt> UniqInChI dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
```

(continues on next page)

(continued from previous page)

```
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Read molecules and write out the unique ones. Two molecules are considered
* identical if their InChIs are identical.
*****************************************************************************/
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class UniqInChI
{
    public static void Main(string[] args)
    {
        if (args.Length != 2)
        {
            OEChem.OEThrow.Usage("UniqInChI <infile> <outfile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
        }
        ProcessFile(ifs, ofs);
    }

    private static void ProcessFile(oemolistream ifs, oemolostream ofs)
    {
        OEGraphMol mol = new OEGraphMol();
        HashSet<string> UniqInChIs = new HashSet<string>();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            string inchi = OEChem.OECreateInChI(mol);
            if (UniqInChIs.Add(inchi))
            {
```

(continues on next page)

(continued from previous page)

```
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }
    }
}
```

See also

- *OECreateInChI* function
- *Write out unique molecules (canonical SMILES)* section

Randomize atoms of molecules

A program that randomizes the order of the atoms within the molecule file. The structure of the molecule is not changed.

Example

```
prompt> RandomizeAtoms dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Demonstrates how to randomly reorder atoms and bonds of a molecule
//****************************************************************************/
using System;
using OpenEye.OEChem;

public class RandomizeAtoms
{

    public static void Main(string[] args)
```

(continues on next page)

(continued from previous page)

```
    {
        if (args.Length != 2)
        {
            OEChem.OEThrow.Usage("RandomizeAtoms <infile> <outfile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEScrambleMolecule(mol);
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

See also

- *OEScrambleMolecule* function

Generate canonical SMILES

A program that generates the canonical SMILES of the molecules in the input file. It has a number of options such as `-from3d`, which perceives stereo from the 3D coordinates, `-isomeric`, which produces the canonical isomeric SMILES, and `-kekule`, which produces the Kekulé SMILES form.

Example

```
prompt> CanSmi dbase.oeb.gz output.smi
prompt> CanSmi -from3d -i dbase.sdf -o output.smi
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
```

(continues on next page)

(continued from previous page)

```
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Generate canonical smiles of various flavors
*****************************************************************************/
using System;
using System.IO;
using OpenEye.OEChem;

/*****************************************************************************
 To create unique Kekule smiles, must reperceive bond orders from
 scratch to avoid arbitrary nondeterministic variations, e.g.,
 CC1=C(O)C=CC=C1 vs. CC1=CC=CC=C1O
 This is why OESMILESFlag_Kekule is not sufficient and not used.
*****************************************************************************/

public class CanSmi
{
    private static string CreateCanSmi(OEGraphMol mol, bool isomeric, bool kekule)
    {
        OEChem.OEFindRingAtomsAndBonds(mol);
        OEChem.OEAssignAromaticFlags(mol, OEAroModel.OpenEye);
        uint smiflag = OESMILESFlag.Canonical;
        if (isomeric)
        {
            smiflag |= OESMILESFlag.ISOMERIC;
        }
        if (kekule)
        {
            foreach (OEBondBase bond in mol.GetBonds(new OEIsAromaticBond()))
            {
                bond.SetIntType(5);
            }
            OEChem.OECanonicalOrderAtoms(mol);
            OEChem.OECanonicalOrderBonds(mol);
            OEChem.OEClearAromaticFlags(mol);
            OEChem.OEKekulize(mol);
        }
        string smi = OEChem.OECreateSmiString(mol, smiflag);
        return smi;
    }
```

(continues on next page)

(continued from previous page)

```
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "CanSmi", args);

        bool isomeric = itf.GetBool("-isomeric");
        bool kekule   = itf.GetBool("-kekule");
        bool from3d   = itf.GetBool("-from3d");

        if (from3d)
        {
            isomeric = true;
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        StreamWriter sw = null;
        if (itf.HasString("-o"))
        {
            try
            {
                sw = new StreamWriter(itf.GetString("-o"));
                Console.SetOut(sw);
            }
            catch
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for␣
↪writing");
            }
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (from3d)
            {
                OEChem.OE3DToInternalStereo(mol);
            }
            string smi = CreateCanSmi(mol, isomeric, kekule);
            string title = mol.GetTitle();
            if (!string.IsNullOrEmpty(title))
            {
                smi += " " + title;
            }
            Console.WriteLine(smi);
        }
        if (itf.HasString("-o"))
        {
```

(continues on next page)

(continued from previous page)

```
            sw.Close();
        }
    }

    private static string interfaceData=@"
!BRIEF [options] [-i] <input> [[-o] <output>]
!PARAMETER -i
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !ALIAS -out
  !TYPE string
  !BRIEF output file name
  !KEYLESS 2
!END
!PARAMETER -isomeric
  !TYPE bool
  !DEFAULT false
  !BRIEF generate isomeric smiles
!END
!PARAMETER -from3d
  !TYPE bool
  !DEFAULT false
  !BRIEF perceive stereo from 3D coords
!END
!PARAMETER -kekule
  !TYPE bool
  !DEFAULT false
  !BRIEF generate kekule form
!END
";
}
```

See also

- *OEFindRingAtomsAndBonds* function
- *OEAssignAromaticFlags* function
- *OEKekulize* function
- *OECanonicalOrderAtoms* function
- *OECanonicalOrderBonds* function

Filter molecules by weight or heavy atom count

A program that filters molecules by their weight and/or heavy atom count. The program has four flags: `-minhac`, the minimum heavy atom count, `-maxhac`, the maximum heavy atom count, `-minwt`, the minimum molecular weight and `-maxwt` the maximum molecular weight. Any combination of the flags can be set. The program will output the molecules to stdout as SMILES if no output file is set.

Example

```
prompt> SizeFilter -minhac 10 dbase.oeb.gz
prompt> SizeFilter -minhac 10 -maxwt 300 -i dbase.oeb.gz -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Filter out molecules by their molecular weight or heavy atom count
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class SizeFilter
{
    public static void Main(string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        OEInterface itf = new OEInterface(interfaceData, "SizeFilter", args);

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        oemolostream ofs = new oemolostream(".ism");
        if (itf.HasString("-o"))
        {
            if (!ofs.open(itf.GetString("-o")))
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for␣
↪writing");
            }
        }

        int minhac = int.MinValue;
        if (itf.HasInt("-minhac"))
        {
            minhac = itf.GetInt("-minhac");
        }

        int maxhac = int.MaxValue;
        if (itf.HasInt("-maxhac"))
        {
            maxhac = itf.GetInt("-maxhac");
        }

        double minwt = double.MinValue;
        if (itf.HasDouble("-minwt"))
        {
            minwt = itf.GetDouble("-minwt");
        }

        double maxwt = double.MaxValue;
        if (itf.HasDouble("-maxwt"))
        {
            maxwt = itf.GetDouble("-maxwt");
        }

        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (!IsMoleculeInHeavyAtomCountRange(minhac, maxhac, mol))
            {
                continue;
            }

            if (!IsMoleculeInMolWtRange(minwt, maxwt, mol))
```

(continues on next page)

(continued from previous page)

```
            {
                continue;
            }

            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    private static bool IsBetween(int min, int max, int val)
    {
        if (val >= min && val <= max)
        {
            return true;
        }
        return false;
    }

    private static bool IsBetween(double min, double max, double val)
    {
        if (val >= min && val <= max)
        {
            return true;
        }
        return false;
    }

    private static bool IsMoleculeInHeavyAtomCountRange(int minhac, int maxhac, OEMol␣
↪mol)
    {
        uint count = OEChem.OECount(mol, new OEIsHeavy());
        return IsBetween(minhac, maxhac, count);
    }

    private static bool IsMoleculeInMolWtRange(double minwt, double maxwt, OEMol mol)
    {
        double molwt = OEChem.OECalculateMolecularWeight(mol);
        return IsBetween(minwt, maxwt, molwt);
    }

    private static string interfaceData=@"
!BRIEF [-minhac <num>] [-maxhac <num>] [-minwt <num>] [-maxwt <num>] [-i] <input> [[-o]
↪<output>]
!PARAMETER -i
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !TYPE string
  !REQUIRED false
  !BRIEF Output file name
```

(continues on next page)

(continued from previous page)

```
  !KEYLESS 2
!END
!PARAMETER -minhac
  !TYPE int
  !REQUIRED false
  !BRIEF minimum heavy atom count
!END
!PARAMETER -maxhac
  !TYPE int
  !REQUIRED false
  !BRIEF maximum heavy atom count
!END
!PARAMETER -minwt
  !TYPE double
  !REQUIRED false
  !BRIEF minimum molecular weight
!END
!PARAMETER -maxwt
  !TYPE double
  !REQUIRED false
  !BRIEF maximum molecular weight
!END
";
}
```

See also

- *OEIsHeavy* functor
- *OECalculateMolecularWeight* function

Strip salts

A program that removes all but the largest molecule from molecules that contain more than one part. If the molecule contains two or more parts that are equal in size it keeps the larger.

Example

```
prompt> StripSalts dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
```

(continues on next page)

(continued from previous page)

```
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Remove salts and/or remove all but the largest molecule
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class StripSalts
{

    public static void Main(string[] args)
    {
        if (args.Length != 2)
        {
            OEChem.OEThrow.Usage("StripSalts <infile> <outfile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
        }

        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEDeleteEverythingExceptTheFirstLargestComponent(mol);
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }
}
```

See also

- *OETheFunctionFormerlyKnownAsStripSalts* function

Extract rings

A program that removes all non-ring atoms from the input molecules. By default double bonded atoms exo to a ring are included as ring atoms. This can be changed by using the flag `-exo`. The program will output the molecules to stdout as SMILES if no output file is set. The molecule name will be appended with "_rings".

Example

```
prompt> RingSubset dbase.oeb.gz
prompt> RingSubset -exo false -i dbase.oeb.gz -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Extracting rings/ring systems from input molecules
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class RingSubset
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "RingSubset", args);

        bool exo_dbl_bonds = itf.GetBool("-exo");

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
```

(continues on next page)

(continued from previous page)

```
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        oemolostream ofs = new oemolostream(".ism");
        if (itf.HasString("-o"))
        {
            if (!ofs.open(itf.GetString("-o")))
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for␣
↪writing");
            }
        }
        RingSubSet(ifs, ofs, exo_dbl_bonds);
    }

    private static void RingSubSet(oemolistream ifs, oemolostream ofs, bool exo)
    {
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEGraphMol submol = new OEGraphMol();
            OEUnaryAtomPred pred = new OEAtomIsInRing();
            if (exo)
            {
                pred = new OEOrAtom(pred, new OEIsNonRingAtomDoubleBondedToRing());
            }
            bool adjustHCount = true;
            OEChem.OESubsetMol(submol, mol, pred, adjustHCount);

            string title = mol.GetTitle() + "_rings";
            submol.SetTitle(title);
            if (submol.NumAtoms() != 0)
            {
                OEChem.OEWriteMolecule(ofs, submol);
            }
        }
    }

    private static string interfaceData=@"
!BRIEF [-exo] [-i] <infile> [[-o] <outfile>]
!PARAMETER -i
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !ALIAS -out
  !TYPE string
```

(continues on next page)

(continued from previous page)

```
  !REQUIRED false
  !BRIEF output file name
  !KEYLESS 2
!END
!PARAMETER -exo
  !TYPE bool
  !DEFAULT true
  !BRIEF Include non-ring atoms double bonded to a ring
!END
";
}
```

See also

- *OEAtomIsInRing* functor
- *OEIsNonRingAtomDoubleBondedToRing* functor

Extract molecule scaffolds

This example is a program that removes all atoms that are neither ring atoms nor atoms in linkers between rings. By default, double bonded atoms exo to a ring are included as ring atoms. This can be changed by using the flag `-exo`.

Example

```
prompt> ExtractScaffold dbase.oeb.gz
prompt> ExtractScaffold -exo false -i dbase.oeb.gz -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Extract the ring scaffold of a molecule
```

(continues on next page)

(continued from previous page)

```
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class ExtractScaffold
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "ExtractScaffold", args);

        bool exo_dbl_bonds = itf.GetBool("-exo");

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(itf.GetString("-o")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for writing
↪");
        }

        OEMol src = new OEMol();
        OEMol dst = new OEMol();
        while (OEChem.OEReadMolecule(ifs, src))
        {
            bool adjustHCount = true;
            if (exo_dbl_bonds)
            {
                OEChem.OESubsetMol(dst, src, new OEOrAtom(new IsInScaffold(), new␣
↪OEIsNonRingAtomDoubleBondedToRing()), adjustHCount);
            }
            else
            {
                OEChem.OESubsetMol(dst, src, new IsInScaffold(), adjustHCount);
            }

            if (dst.IsValid())
            {
                OEChem.OEWriteMolecule(ofs, dst);
            }
        }
    }

    private static bool TraverseForRing(bool[] visited, OEAtomBase atom)
    {
        visited[atom.GetIdx()] = true;
```

(continues on next page)

(continued from previous page)

```
        foreach (OEAtomBase nbor in atom.GetAtoms())
        {
            if (!visited[nbor.GetIdx()])
            {
                if (nbor.IsInRing())
                {
                    return true;
                }

                if (TraverseForRing(visited, nbor))
                {
                    return true;
                }
            }
        }

        return false;
    }

    private static bool DepthFirstSearchForRing(OEAtomBase root, OEAtomBase nbor)
    {
        bool[] visited = new bool[root.GetParent().GetMaxAtomIdx()];

        visited[root.GetIdx()] = true;

        return TraverseForRing(visited, nbor);
    }

    private class IsInScaffold : OEUnaryAtomPred
    {
        public override bool Call(OEAtomBase atom)
        {
            if (atom.IsInRing())
            {
                return true;
            }

            int count = 0;
            foreach (OEAtomBase nbor in atom.GetAtoms())
            {
                if (DepthFirstSearchForRing(atom, nbor))
                {
                    ++count;
                }
            }

            return count > 1;
        }

        public override OEUnaryAtomBoolFunc CreateCopy()
        {
            OEUnaryAtomBoolFunc copy = new IsInScaffold();
```

(continues on next page)

(continued from previous page)

```
            copy.ReleaseOwnership();
            return copy;
        }
    }

    private static string interfaceData=@"
!BRIEF [-exo] [-i] <input> [-o] <scaffolds>
!PARAMETER -i
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !ALIAS -out
  !TYPE string
  !REQUIRED true
  !BRIEF output file name
  !KEYLESS 2
!END
!PARAMETER -exo
  !TYPE bool
  !DEFAULT true
  !BRIEF Exclude double bonds exo to ring in scaffold
!END
";
}
```

See also

- *OESubsetMol* function
- *OEAtomIsInRing* functor
- *OEIsNonRingAtomDoubleBondedToRing* functor

Extract random molecule subset

A program that extracts a randomized subset of molecules from a molecule file. The program has two flags, `-p`, extract a percentage of the database, or `-n`, extract a specific number from the database. If neither flag is set then the whole database will be randomized.

Example

```
prompt> RandomSample -n 1000 dbase.oeb.gz output.oeb.gz
prompt> RandomSample -p 10 -i dbase.oeb.gz -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Randomly reorder molecules and optionally obtain a random subset
*****************************************************************************/
using System;
using System.Collections;
using System.Collections.Generic;
using ExtensionMethods;
using OpenEye.OEChem;

namespace ExtensionMethods
{
public static class MyExtensions
{
    public static void Shuffle(this ArrayList list, Random r)
    {

        int n = list.Count;
        while (n > 1)
        {
            n--;
            int k = r.Next(n + 1);
            OEMol value = (OEMol)list[k];
            list[k] = list[n];
            list[n] = value;
        }
    }
}
}

public class RandomSample
{

    public static void LoadDatabase(oemolistream ifs, ArrayList mlist, int count)
```

(continues on next page)

(continued from previous page)

```
    {
        int readcount = 0;
        OEMol mol = new OEMol(OEMCMolType.OEDBMCMol);
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            readcount++;
            mol.Compress();
            mlist.Add(mol);
            mol = new OEMol();
            if (readcount == count)
            {
                break;
            }
        }
    }

    public static void WriteDatabase(oemolostream ofs, ArrayList mlist, int count)
    {
        int outcount = 0;
        foreach (OEMol dbmol in mlist)
        {
            dbmol.UnCompress();
            OEChem.OEWriteMolecule(ofs, dbmol);
            outcount++;
            if (outcount == count)
            {
                break;
            }
        }
    }

    public static void RandomizePercent(oemolistream ifs, oemolostream ofs, float␣
↪percent, Random r)
    {
        ArrayList mlist = new ArrayList();
        LoadDatabase(ifs, mlist, 0);

        mlist.Shuffle(r);

        int size = mlist.Count;
        int count = (int)(percent * 0.01 * size);
        if (count < 1) {
            count = 1;
        }
        WriteDatabase(ofs, mlist, count);
    }

    public static void Randomize(oemolistream ifs, oemolostream ofs, Random r)
    {
        float wholedb = 100;
        RandomizePercent(ifs, ofs, wholedb, r);
    }
```

(continues on next page)

(continued from previous page)

```
    public static void RandomizeN(oemolistream ifs, oemolostream ofs, int count, Random␣
↪r)
    {
        ArrayList mlist = new ArrayList();
        LoadDatabase(ifs, mlist, count);

        int readcount = 0;
        OEMol mol = new OEMol(OEMCMolType.OEDBMCMol);
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if ((float)count/(count + readcount + 1) > r.NextDouble())
            {
                int idx = (int) ((float)count * r.NextDouble());
                mol.Compress();
                mlist[idx] = mol;
                mol = new OEMol();
            }
            readcount++;
        }

        mlist.Shuffle(r);
        WriteDatabase(ofs, mlist, count);
    }

    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "RandomSample", args);

        if (itf.HasFloat("-p") && itf.HasInt("-n"))
        {
            OEChem.OEThrow.Usage("Give only one option, -p or -n");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        oemolostream ofs = new oemolostream(".ism");
        if (itf.HasString("-o"))
        {
            if (!ofs.open(itf.GetString("-o")))
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for␣
↪writing");
            }
        }

        Random r = null;
```

(continues on next page)

(continued from previous page)

```
        if (itf.HasInt("-seed")) {
            r = new Random(itf.GetInt("-seed"));
        }
        else {
            r = new Random();
        }

        if (itf.HasInt("-n"))
        {
            RandomizeN(ifs, ofs, itf.GetInt("-n"), r);
        }
        else if (itf.HasFloat("-p"))
        {
            RandomizePercent(ifs, ofs, itf.GetFloat("-p"), r);
        }
        else
        {
            Randomize(ifs, ofs, r);
        }
    }

    private static string interfaceData=@"
!BRIEF [-seed <int>] [-n <number>] [-p <percent>] [-i] <input> [-o] <output>
!PARAMETER -i
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !TYPE string
  !REQUIRED false
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -p
  !TYPE float
  !REQUIRED false
  !BRIEF Percentage of output molecules
!END
!PARAMETER -n
  !TYPE int
  !REQUIRED false
  !BRIEF Number of output molecules
!END
!PARAMETER -seed
  !TYPE int
  !REQUIRED false
  !BRIEF Integer value for random seed, default is system time
!END
";
}
```

See also

- *OEMolBase.Compress* method
- *OEMolBase.UnCompress* method
- Manipulating Large Molecule Files recipe in Python cookbook.

Split molecule file

A program that splits molecule file into N chunks or chunks of size N. Number of chunks (`-num`) or the size of each chunk (`-size`) must be specified and are mutually exclusive. By default the program splits molecule file by number of conformers not molecules. This can be changed by setting the flag `-confs` to false.

Example

```
prompt> MolChunk -num 3 -i drugs.sdf -o foo.sdf
prompt> MolChunk -confs false -size 20 -i conftest.sdf -o foo.sdf
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Split molecule file into N chunks or chunks of size N
//****************************************************************************/
using System;

using OpenEye.OEChem;

public class MolChunk
{
    public static void NewOutputStream(oemolostream ofs, string outbase, string ext,
                                       uint chunk)
    {
        string newname = outbase + "_" + chunk.ToString().PadLeft(7, '0') + "." + ext;
```

(continues on next page)

(continued from previous page)

```
        if (!ofs.open(newname))
        {
            OEChem.OEThrow.Fatal("Unable to open " + newname + " for writing");
        }
    }

    public static void SplitNParts(oemolistream ifs, uint nparts, bool countconfs,
                                   string outbase, string ext)
    {
        uint molconfcount = 0;
        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (countconfs)
            {
                molconfcount += mol.NumConfs();
            }
            else
            {
                molconfcount += 1;
            }
        }
        ifs.rewind();

        uint chunksize = (uint)(molconfcount / nparts);
        uint lft = (uint)(molconfcount % nparts);
        if (lft != 0)
        {
            chunksize += 1;
        }
        uint chunk = 1;
        uint count = 0;

        oemolostream ofs = new oemolostream();
        NewOutputStream(ofs, outbase, ext, chunk);
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (countconfs)
            {
                count += mol.NumConfs();
            }
            else
            {
                count += 1;
            }
            if (count > chunksize)
            {
                if (chunk == lft)
                {
                    chunksize -= 1;
                }
                ofs.close();
```

(continues on next page)

(continued from previous page)

```
                chunk += 1;
                count = 1;
                NewOutputStream(ofs, outbase, ext, chunk);
            }
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    public static void SplitChunk(oemolistream ifs, uint chunksize, bool countconfs,
                                  string outbase, string ext)
    {
        uint chunk = 1;
        oemolostream ofs = new oemolostream();
        NewOutputStream(ofs, outbase, ext, chunk);

        uint count = 0;
        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (count >= chunksize)
            {
                ofs.close();
                count = 0;
                chunk += 1;
                NewOutputStream(ofs, outbase, ext, chunk);
            }
            if (countconfs)
            {
                count += mol.NumConfs();
            }
            else
            {
                count += 1;
            }
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "MolChunk", args);

        if (!(itf.HasInt("-num") ^ itf.HasInt("-size")))
        {
            OEChem.OEThrow.Fatal("Number of chunks (-num) or the size of each chunk (-
↪size) must be specified and are mutually exclusive.");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
```

(continues on next page)

(continued from previous page)

```
↪");
        }

        uint fmt = ifs.GetFormat();
        if (fmt != OEFormat.OEB)
        {
            ifs.SetConfTest(new OEIsomericConfTest(false));
        }
        else if (itf.GetBool("-rotcompress"))
        {
            OEChem.OEPreserveRotCompress(ifs);  // only for OEB
        }

        string ofname = itf.GetString("-o");
        string outbase = null;
        string ext = OEChem.OEGetFileExtension(ofname);
        if (ext != null) {
            int place = ofname.LastIndexOf(ext);
            outbase = ofname.Substring(0, place-1);
        }
        else
        {
            OEChem.OEThrow.Fatal("Failed to find file extension");
        }

        bool countconfs = itf.GetBool("-confs");

        if (itf.HasInt("-num"))
        {
            uint nparts = (uint)itf.GetInt("-num");
            SplitNParts(ifs, nparts, countconfs, outbase, ext);
        }
        else
        {
            uint chunksize = (uint)itf.GetInt("-size");
            SplitChunk(ifs, chunksize, countconfs, outbase, ext);
        }
    }

    private static string interfaceData=@"
!BRIEF -num|-size [-i] <input> [-o] <output>
!PARAMETER -i 1
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -o 2
  !TYPE string
  !REQUIRED true
  !BRIEF Output file name
  !KEYLESS 2
```

(continues on next page)

(continued from previous page)

```
!END
!PARAMETER -num 3
  !TYPE int
  !BRIEF The number of chunks
!END
!PARAMETER -size 4
  !TYPE int
  !BRIEF The size of each chunk
!END
!PARAMETER -confs 5
  !TYPE bool
  !DEFAULT true
  !BRIEF Split by number of conformers not molecules
!END
!PARAMETER -rotcompress 6
  !TYPE bool
  !DEFAULT false
  !BRIEF Preserve rotor compression
!END
";
}
```

See also

- *OEIsomericConfTest* class

Performing a reaction

A program that uses a SMIRKS to perform reactions on single molecules. The OpenEye OEUniMolecularRxn class is designed to react every instance of the reactant pattern in the input molecules and so is useful for normalization reactions.

Example

```
prompt> UniMolRxn '[NH2:1]>>[Nh3+:1]' dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
```

(continues on next page)

(continued from previous page)

```
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Perform reactions on the given compounds
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class UniMolRxn
{
    public static void Main(string[] args)
    {
        if (args.Length < 2 || args.Length > 3)
        {
            OEChem.OEThrow.Usage("UniMolRxn SMIRKS <infile> [<outfile>]");
        }

        OEQMol qmol = new OEQMol();
        if (!OEChem.OEParseSmirks(qmol, args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to parse SMIRKS: " + args[0]);
        }

        OEUniMolecularRxn umr = new OEUniMolecularRxn();
        if (!umr.Init(qmol))
        {
            OEChem.OEThrow.Fatal("Failed to initialize reaction with " + args[0] + "␣
↪SMIRKS");
        }
        umr.SetClearCoordinates(true);

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for reading");
        }

        oemolostream ofs = new oemolostream(".ism");
        if (args.Length == 3)
        {
            if (!ofs.open(args[2]))
            {
                OEChem.OEThrow.Fatal("Unable to open " + args[2] + " for writing");
            }

        }
        UniMolecularRxn(ifs, ofs, umr);
    }
```

(continues on next page)

(continued from previous page)

```
    private static void UniMolecularRxn(oemolistream ifs,
                                        oemolostream ofs,
                                        OEUniMolecularRxn umr)
    {

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (umr.Call(mol))
            {
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }
    }
}
```

See also

- *OEUniMolecularRxn* class
- *Library generation* section

Library generation

A program that uses a SMIRKS string or an MDL reaction file to perform reactions on input molecules. The program has a number of flags. OELibraryGen options are: `-implicitH`, performs the reaction with implicit hydrogens (default `false`), `-relax`, ensures unmapped atoms on the reactant side are not deleted during reaction (default `false`), `-valence`, applies automatic valence correction (default `false`). SMILES generation options are: `-isomeric`, includes atom and bond stereochemistry in the output (default `false`), `-unique`, only include unique product canonical SMILES (default `false`). Input and output options are: `-reactants`, a molecule file, or files, of reactants, `-smirks`, a SMIRKS string of the reaction or `-rxn`, an MDL reaction file of the reaction, and optionally, `-product`, the output molecule file. The program will output the molecules to stdout as SMILES if no output file is set.

Example

```
prompt > LibGen -smirks '[C:1]c1c([N:2])cccc1>>[C:1]c1cc([N:2])ccc1' -reactants input.smi
prompt > LibGen -rxn reaction.rxn -reactants input.smi -product output.smi -valence -
↪isomeric
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
```

(continues on next page)

(continued from previous page)

```
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Perform library generation with SMIRKS
*****************************************************************************/
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class LibGen
{
    private static void LibraryGen(OELibraryGen libgen, oemolostream ofs,
                                   bool isomeric, bool unique)
    {
        uint smiflag = OESMILESFlag.DEFAULT;
        if (isomeric)
        {
            smiflag |= OESMILESFlag.ISOMERIC;
        }
        HashSet<string> uniqueproducts = new HashSet<string>();
        foreach (OEMolBase mol in libgen.GetProducts())
        {
            string smiles = OEChem.OECreateSmiString(mol, smiflag);
            if (!unique || !uniqueproducts.Contains(smiles))
            {
                uniqueproducts.Add(smiles);
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }
    }

    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "LibGen", args);
        if (!(itf.HasString("-smirks") ^ itf.HasString("-rxn")))
        {
            OEChem.OEThrow.Fatal("Please provide SMIRKS string or MDL reaction file");
        }

        OEQMol reaction = new OEQMol();
        if (itf.HasString("-smirks"))
        {
            string smirks = itf.GetString("-smirks");
            if (!OEChem.OEParseSmirks(reaction, smirks))
```

(continues on next page)

(continued from previous page)

```
            {
                OEChem.OEThrow.Fatal("Unable to parse SMIRKS: " + smirks);
            }
        }
        else
        {
            string rxn = itf.GetString("-rxn");
            oemolistream rfile = new oemolistream(rxn);
            uint opt = OEMDLQueryOpts.ReactionQuery|OEMDLQueryOpts.SuppressExplicitH;
            if (!OEChem.OEReadMDLReactionQueryFile(rfile, reaction, opt))
            {
                OEChem.OEThrow.Fatal("Unable to read reaction file: " + rxn);
            }
        }
        bool relax      = itf.GetBool("-relax");
        bool unique     = itf.GetBool("-unique");
        bool implicitH  = itf.GetBool("-implicitH");
        bool valcorrect = itf.GetBool("-valence");
        bool isomeric   = itf.GetBool("-isomeric");

        OELibraryGen libgen = new OELibraryGen();

        // Initialize library generation
        if (!libgen.Init(reaction, !relax))
        {
            OEChem.OEThrow.Fatal("failed to initialize library generator");
        }
        libgen.SetValenceCorrection(valcorrect);
        libgen.SetExplicitHydrogens(!implicitH);
        libgen.SetClearCoordinates(true);

        uint nrReacts = 0;
        foreach (string filename in itf.GetStringList("-reactants"))
        {
            if (nrReacts >= libgen.NumReactants())
            {
                OEChem.OEThrow.Fatal("Number of reactant files exceeds number of␣
↪reactants specified in reaction");
            }
            oemolistream ifs = new oemolistream();
            if (!ifs.open(filename))
            {
                OEChem.OEThrow.Fatal("Unable to open " + filename + " for reading");
            }
            OEGraphMol mol = new OEGraphMol();
            while (OEChem.OEReadMolecule(ifs, mol))
            {
                libgen.AddStartingMaterial(mol, nrReacts, unique);
            }
            nrReacts++;
        }
        if (nrReacts != libgen.NumReactants())
```

(continues on next page)

(continued from previous page)

```
        {
            OEChem.OEThrow.Fatal("Reactions requires " + libgen.NumReactants() + "␣
↪reactant files!");
        }

        oemolostream ofs = new oemolostream(".ism");
        if (itf.HasString("-product"))
        {
            if (!ofs.open(itf.GetString("-product")))
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-product") + "␣
↪for writing");
            }
        }
        LibraryGen(libgen, ofs, isomeric, unique);
    }

    private static string interfaceData=@"
!BRIEF [options] [-smirks <string> | -rxn <rfile>] -reactants <infile> [-product
↪<outfile>]
!CATEGORY ""input/output options""
!PARAMETER -smirks
!ALIAS -s
!TYPE string
!VISIBILITY simple
!BRIEF SMIRKS reaction string
!END

!PARAMETER -rxn
!ALIAS -s
!TYPE string
!VISIBILITY simple
!BRIEF MDL reaction file
!END

!PARAMETER -reactants
!ALIAS -r
!TYPE string
!LIST true
!REQUIRED true
!VISIBILITY simple
!BRIEF list of input reactant filenames
!END

!PARAMETER -product
!ALIAS -p
!TYPE string
!REQUIRED false
!VISIBILITY simple
!BRIEF output product filename
!END
!END
```

(continues on next page)

(continued from previous page)

```
!CATEGORY ""OELibraryGen options""
!PARAMETER -relax
!TYPE bool
!REQUIRED false
!DEFAULT false
!VISIBILITY simple
!BRIEF unmapped atoms on reactant side are not deleted during reaction
!END

!PARAMETER -implicitH
!TYPE bool
!REQUIRED false
!DEFAULT false
!VISIBILITY simple
!BRIEF reaction will be perfomed using implicit hydrogens
!END

!PARAMETER -valence
!TYPE bool
!REQUIRED false
!DEFAULT false
!VISIBILITY simple
!BRIEF automatic valence correction will be applied
!END
!END

!CATEGORY ""product smiles generation options""
!PARAMETER -unique
!TYPE bool
!REQUIRED false
!DEFAULT false
!VISIBILITY simple
!BRIEF only unique product canonical smiles will be written
!END

!PARAMETER -isomeric
!TYPE bool
!REQUIRED false
!DEFAULT false
!VISIBILITY simple
!BRIEF include atom and bond stereochemistry in product smiles string
!END
!END
";
}
```

See also

- *OELibraryGen* class
- *OEParseSmirks* function

- *OEReadMDLReactionQueryFile* function

Perform substructure searches

A program that searches a molecule file using a SMARTS pattern. Flags include: `-c`, count the number of matches, `-o`, output the matches to a file, `-p`, the SMARTS pattern to use, `-r`, do the reverse search `i.e.` all molecules that don't match the SMARTS.

Example

```
prompt> MolGrep -p 'c1ccccc1' -c dbase.oeb.gz
prompt> MolGrep -r -p 'c1ccccc1' -i dbase.oeb.gz -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Perform substructure search on molecule file
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class MolGrep
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "MolGrep", args);
        if (!(itf.GetBool("-c") ^ itf.HasString("-o")))
        {
            OEChem.OEThrow.Fatal("Counting (-c) or output (-o) must be specified and are␣
↪mutually exclusive.");
        }

        oemolistream ifs = new oemolistream();
```

(continues on next page)

(continued from previous page)

```
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        oemolostream ofs = new oemolostream();
        if (!itf.GetBool("-c"))
        {
            if (!ofs.open(itf.GetString("-o")))
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for␣
↪writing");
            }
        }

        string smarts = itf.GetString("-p");
        OESubSearch ss = new OESubSearch();
        if (!ss.Init(smarts))
        {
            OEChem.OEThrow.Fatal("Unable to parse SMARTS: " + smarts);
        }

        SubSearch(itf, ss, ifs, ofs);

    }

    public static void SubSearch(OEInterface itf, OESubSearch ss,
                                 oemolistream ifs, oemolostream ofs)
    {
        bool reverseflag = itf.GetBool("-r");
        bool countflag   = itf.GetBool("-c");
        int count = 0;
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEPrepareSearch(mol, ss);
            if (ss.SingleMatch(mol) != reverseflag)
            {
                if (countflag)
                {
                    count++;
                }
                else
                {
                    OEChem.OEWriteMolecule(ofs, mol);
                }
            }
        }
        if (countflag)
        {
            Console.WriteLine(count + " matching molecules");
```

(continues on next page)

(continued from previous page)

```
        }
    }

    private static string interfaceData=@"
!BRIEF [-r][-c] -p smarts [-i] <input> [[-o] <output>]
!PARAMETER -i 1
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -p 2
  !TYPE string
  !REQUIRED true
  !BRIEF SMARTS pattern, quote for safety
!END
!PARAMETER -o 3
  !ALIAS -out
  !TYPE string
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -r 4
  !ALIAS -v
  !TYPE bool
  !DEFAULT false
  !BRIEF Reverse logic, not matched
!END
!PARAMETER -c 5
  !TYPE bool
  !DEFAULT false
  !BRIEF Just output count of number matched
!END
";
}
```

See also

- *OESubSearch* class
- *OESubSearch.SingleMatch* method
- *OEPrepareSearch* function

Generate database for fast substructure search

A program that generates a binary file for fast substructure search. For more explanation, see *OECreateSubSearchDatabaseFile* function.

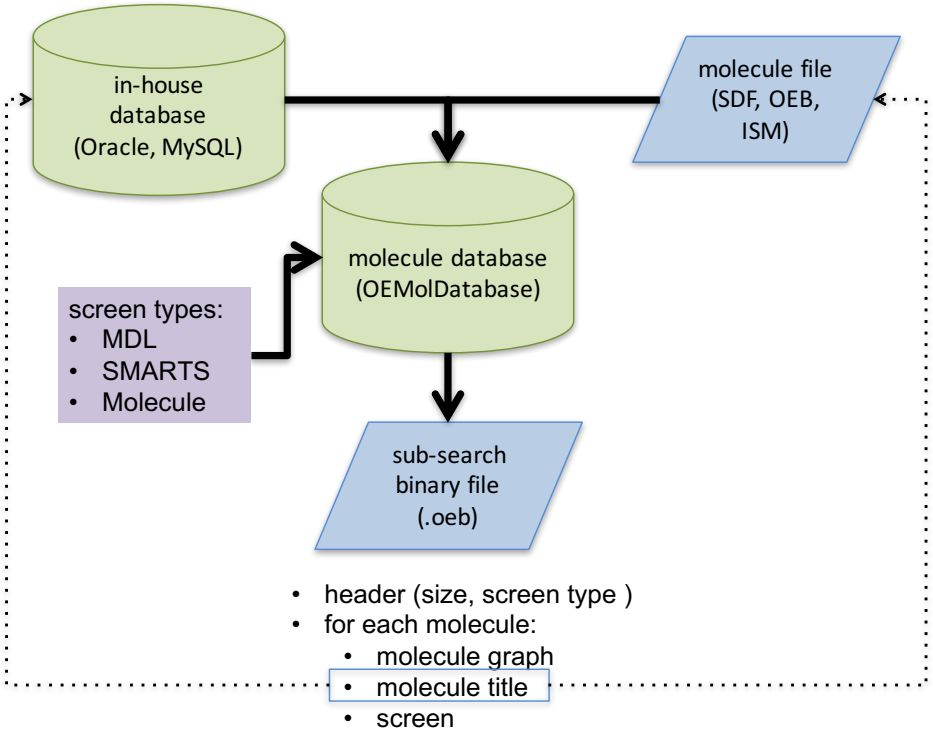


Fig. 1: **Schematic representation of substructure search database file generation process**

Code Example

- *Perform SMARTS search on substructure database file* example

Command Line Interface

A description of the command line interface can be obtained by executing the program with the –help argument.

will generate the following output:

```
Simple parameter list
 input/output options
   -in : Input molecule filename
```

(continues on next page)

(continued from previous page)

```
  -out : Output substructure database filename

other options
  -keep-title : Whether to keep the title of the molecule as unique
                identifier
  -nrthreads : Number of processors used (zero means all available)
  -screentype : Screen type generated
  -sort : Whether to sort the molecules based on their screen bit counts.
```

Example

Code

See also

- *OECreateSubSearchDatabaseOptions* class
- *OESubSearchScreenTypeBase* class
- *OEConsoleProgressTracer* class
- `OEGetSubSearchScreenType` function
- `OECreateSubSearchDatabaseFile` function
- `OECreateSubSearchDatabaseOptions` function

Perform SMARTS search on substructure database file

A program that searches substructure search database using a SMARTS pattern. For more explanation, see `OESubSearchDatabase` class.

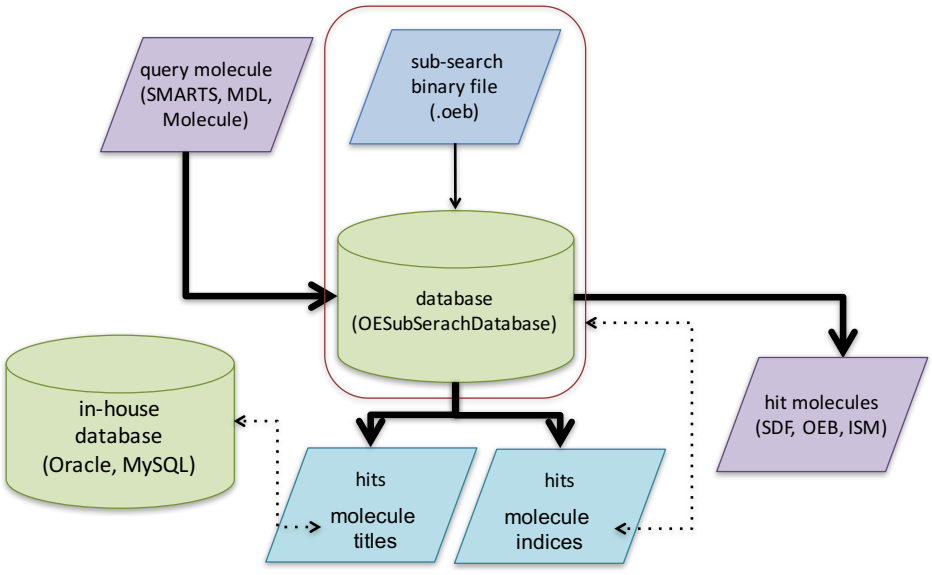


Fig. 2: **Schematic representation of using substructure search database**

Code Example

- *Generate database for fast substructure search* example

Command Line Interface

A description of the command line interface can be obtained by executing the program with the –help argument.

will generate the following output:

```
Simple parameter list
 input/output options
   -database : Input substructure search database filename
   -smarts : SMARTS pattern, quote for safety

 other options
   -maxmatches : The maximum number of matches returned in case of parameters
                 (-titles) and (-out)
   -nrthreads : Number of processors used (zero means all available)

 search types
   -count : Output count of number matched
   -out : Output molecule filename of matches
   -titles : Output title of matches
```

Example

Code

See also

- *OESubSearchDatabase* class
- *OESubSearchQuery* class
- *OESubSearchResult* class
- *OEConsoleProgressTracer* class
- *OEGetSubSearchScreenType* function
- *OEIsValidSubSearchDatabase* function
- *OESubSearchStatusToName* function

Align molecules by maximum common substructure

A program that aligns a database of molecules with a reference molecule based on the maximum common substructure (MCS) between the reference and each query. The output file consists of the input reference molecule and then each aligned database molecule.

Example

```
prompt > MCS3DAlign ligand.pdb dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Align two compounds based on the maximum common substructure
//****************************************************************************/
using System;
using OpenEye.OEChem;

public class MCS3DAlign {
    public static void MCSAlign(OEMolBase refmol, OEMolBase fitmol,
                                oemolostream ofs)
    {
        uint atomexpr = OEExprOpts.AtomicNumber|OEExprOpts.Aromaticity;
        uint bondexpr =  0; // ignore bond order
        uint mcsstype = OEMCSType.Exhaustive;
        OEMCSSearch mcss = new OEMCSSearch(refmol, atomexpr, bondexpr, mcsstype);
        mcss.SetMCSFunc(new OEMCSMaxBondsCompleteCycles());

        bool unique = true;
        foreach (OEMatchBase match in mcss.Match(fitmol, unique))
        {
            double[] rmat = new double[9];
            double[] trans = new double[3];
            bool overlay = true;
            double rms = OEChem.OERMSD(mcss.GetPattern(), fitmol, match, overlay, rmat,␣
↪trans);
            if (rms < 0.0)
            {
                OEChem.OEThrow.Warning("RMS overlay failure");
                continue;
```

(continues on next page)

(continued from previous page)

```
            }
            OEChem.OERotate(fitmol, rmat);
            OEChem.OETranslate(fitmol, trans);
            OEChem.OEWriteMolecule(ofs, fitmol);
        }
    }

    public static bool Is3DFormat(uint fmt)
    {
        if (fmt == OEFormat.SMI || fmt == OEFormat.ISM ||
            fmt == OEFormat.CAN || fmt == OEFormat.MF)
        {
            return false;
        }
        return true;
    }

    public static void Main(string[] args)
    {
        if (args.Length != 3)
        {
            OEChem.OEThrow.Usage("MCS3DAlign <refmol> <fitmol> <outfile>");
        }

        oemolistream reffs = new oemolistream();
        if (!reffs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }
        if (!Is3DFormat(reffs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input format: need 3D coordinates");
        }
        OEGraphMol refmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(reffs, refmol))
        {
            OEChem.OEThrow.Fatal("Unable to read molecule in " + args[0]);
        }
        if (refmol.GetDimension() != 3)
        {
            OEChem.OEThrow.Fatal(refmol.GetTitle() + " doesn't have 3D coordinates");
        }

        oemolistream fitfs = new oemolistream();
        if (!fitfs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for reading");
        }
        if (!Is3DFormat(fitfs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input format: need 3D coordinates");
        }
```

(continues on next page)

(continued from previous page)

```
        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[2]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[2] + " for writing");
        }
        if (!Is3DFormat(ofs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid output format: need 3D coordinates");
        }

        OEChem.OEWriteConstMolecule(ofs, refmol);
        OEChem.OESuppressHydrogens(refmol);

        OEGraphMol fitmol = new OEGraphMol();
        while (OEChem.OEReadMolecule(fitfs, fitmol))
        {
            if (fitmol.GetDimension() != 3)
            {
                OEChem.OEThrow.Warning(fitmol.GetTitle() + " doesn't have 3D coordinates
↪");
                continue;
            }
            MCSAlign(refmol, fitmol, ofs);
        }
    }
}
```

See also

- *OEMCSSearch* class
- *OERotate* function
- *OETranslate* function

Align molecules by clique match

A program that aligns a database of molecules with a reference molecule based on the clique of matches between the reference and each query. Clique detection is the process of finding all possible correspondences between two graphs within a set of bounds, the upper bound being the MCS, and in the this example, the lower bound is up to five atoms different to the MCS.

Example

```
prompt > CliqueAlign ligand.pdb dbase.oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Align two compounds based on the clique match
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class CliqueAlign
{
    private static void CliqueAlignMols(OEMolBase refmol, OEMolBase fitmol,
                                        oemolostream ofs)
    {
        uint atomexpr = OEExprOpts.DefaultAtoms;
        uint bondexpr = OEExprOpts.DefaultBonds;
        OECliqueSearch cs = new OECliqueSearch(refmol, atomexpr, bondexpr);
        cs.SetSaveRange(5);
        cs.SetMinAtoms(6);
        foreach (OEMatchBase match in cs.Match(fitmol))
        {
            double[] rmat = new double[9];
            double[] trans = new double[3];
            bool overlay = true;
            OEChem.OERMSD(cs.GetPattern(), fitmol, match, overlay, rmat, trans);
            OEChem.OERotate(fitmol, rmat);
            OEChem.OETranslate(fitmol, trans);
            OEChem.OEWriteMolecule(ofs, fitmol);
        }
    }

    public static bool Is3DFormat(uint fmt)
    {
        if (fmt == OEFormat.SMI || fmt == OEFormat.ISM ||
            fmt == OEFormat.CAN || fmt == OEFormat.MF)
        {
```

(continues on next page)

(continued from previous page)

```
            return false;
        }
        return true;
    }

    public static void Main(string[] args)
    {
        if (args.Length != 3)
        {
            OEChem.OEThrow.Usage("CliqueAlign <refmol> <fitmol> <outfile>");
        }

        oemolistream reffs = new oemolistream();
        if (!reffs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }
        if (!Is3DFormat(reffs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input format: need 3D coordinates");
        }
        OEGraphMol refmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(reffs, refmol))
        {
            OEChem.OEThrow.Fatal("Unable to read molecule in " + args[0]);
        }
        if (refmol.GetDimension() != 3)
        {
            OEChem.OEThrow.Fatal(refmol.GetTitle() + " doesn't have 3D coordinates");
        }

        oemolistream fitfs = new oemolistream();
        if (!fitfs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for reading");
        }
        if (!Is3DFormat(fitfs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input format: need 3D coordinates");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[2]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[2] + " for writing");
        }
        if (!Is3DFormat(ofs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid output format: need 3D coordinates");
        }

        OEChem.OEWriteConstMolecule(ofs, refmol);
```

(continues on next page)

(continued from previous page)

```
        OEChem.OESuppressHydrogens(refmol);

        OEGraphMol fitmol = new OEGraphMol();
        while (OEChem.OEReadMolecule(fitfs, fitmol))
        {
            if (fitmol.GetDimension() != 3)
            {
                OEChem.OEThrow.Warning(fitmol.GetTitle() + " doesn't have 3D coordinates
↪");
                continue;
            }
            CliqueAlignMols(refmol, fitmol, ofs);
        }

    }

}
```

See also

- *OECliqueSearch* class
- *OERMSD* function
- *OEWriteConstMolecule* function
- *OESuppressHydrogens* function

Align molecules by SMARTS match

A program that aligns a database of molecules with a reference molecule based on the SMARTS matches between the reference and each query.

Example

```
prompt > SMARTSAlign ligand.pdb dbase.oeb.gz output.oeb.gz 'a1aaaa1NC'
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
```

(continues on next page)

(continued from previous page)

```
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Align two compounds based on smarts match
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class SMARTSAlign
{
    public static void SmartsAlign(OEMolBase refmol, OEMolBase fitmol,
                                   OESubSearch ss, oemolostream ofs)
    {
        bool unique = true;
        foreach (OEMatchBase match1 in ss.Match(refmol, unique))
        {
            foreach (OEMatchBase match2 in ss.Match(fitmol, unique))
            {
                OEMatch match = new OEMatch();
                OEMatchPairAtomIter mbi1 = match1.GetAtoms();
                OEMatchPairAtomIter mbi2 = match2.GetAtoms();
                while (mbi1.IsValid() && mbi2.IsValid())
                {
                    match.AddPair(mbi1.Target().target, mbi2.Target().target);
                    mbi1.Increment();
                    mbi2.Increment();
                }
                bool overlay = true;
                double[] rmat = new double[9];
                double[] trans = new double[3];
                OEChem.OERMSD(refmol, fitmol, match, overlay, rmat, trans);
                OEChem.OERotate(fitmol, rmat);
                OEChem.OETranslate(fitmol, trans);
                OEChem.OEWriteConstMolecule(ofs, fitmol);
            }
        }
    }

    public static void Main(string[] args)
    {
        if (args.Length != 4)
        {
            OEChem.OEThrow.Usage("SMARTSAlign <refmol> <fitmol> <outfile> <smarts>");
        }

        oemolistream reffs = new oemolistream();
        if (!reffs.open(args[0]))
        {
```

(continues on next page)

(continued from previous page)

```
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }
        if (!OEChem.OEIs3DFormat(reffs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input format: need 3D coordinates");
        }
        OEGraphMol refmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(reffs, refmol))
        {
            OEChem.OEThrow.Fatal("Unable to read molecule in " + args[0]);
        }
        if (refmol.GetDimension() != 3)
        {
            OEChem.OEThrow.Fatal(refmol.GetTitle() + " doesn't have 3D coordinates");
        }

        oemolistream fitfs = new oemolistream();
        if (!fitfs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for reading");
        }
        if (!OEChem.OEIs3DFormat(fitfs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input format: need 3D coordinates");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[2]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[2] + " for writing");
        }
        if (!OEChem.OEIs3DFormat(ofs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid output format: need 3D coordinates");
        }

        OEChem.OEWriteConstMolecule(ofs, refmol);

        OESubSearch ss = new OESubSearch();
        if (!ss.Init(args[3]))
        {
            OEChem.OEThrow.Fatal("Unable to parse SMARTS: " + args[3]);
        }

        OEChem.OEPrepareSearch(refmol, ss);
        if (!ss.SingleMatch(refmol))
        {
            OEChem.OEThrow.Fatal("SMARTS fails to match refmol");
        }

        OEGraphMol fitmol = new OEGraphMol();
        while (OEChem.OEReadMolecule(fitfs, fitmol))
```

(continues on next page)

(continued from previous page)

```
        {
            if (fitmol.GetDimension() != 3)
            {
                OEChem.OEThrow.Warning(fitmol.GetTitle() + " doesn't have 3D coordinates
↪");
                continue;
            }
            OEChem.OEPrepareSearch(fitmol, ss);
            if (!ss.SingleMatch(fitmol))
            {
                OEChem.OEThrow.Warning("SMARTS fails to match fitmol " + fitmol.
↪GetTitle());
                continue;
            }

            SmartsAlign(refmol, fitmol, ss, ofs);
        }
    }
}
```

See also

- *OERMSD* function
- *OEMatch.AddPair* method
- *OEIs3DFormat* function
- *OEPrepareSearch* function

Align multi-conformer molecules

A program that performs RMSD calculation between a 3D reference molecule and multi-conformation molecules.

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
```

(continues on next page)

(continued from previous page)

```
or its use.
*/
/*****************************************************************************
 * Performing RMSD calculation between a 3D reference molecule and
 * multi-conformer molecules
 *****************************************************************************/
using OpenEye.OEChem;
using System;

public class RMSD
{
    private static void RMSDUtility(OEInterface itf)
    {
        if (!itf.GetBool("-verbose"))
        {
            OEChem.OEThrow.SetLevel(OEErrorLevel.Warning);
        }

        string rfname = itf.GetString("-ref");
        string ifname = itf.GetString("-in");

        bool automorph = itf.GetBool("-automorph");
        bool heavy = itf.GetBool("-heavyonly");
        bool overlay = itf.GetBool("-overlay");

        oemolistream refifs = new oemolistream();
        if (!refifs.open(rfname))
        {
            OEChem.OEThrow.Fatal("Unable to open " + rfname + " for reading");
        }

        OEGraphMol rmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(refifs, rmol))
        {
            OEChem.OEThrow.Fatal("Unable to read reference molecule");
        }
        refifs.close();

        oemolistream ifs = new oemolistream();
        if (!ifs.open(ifname))
        {
            OEChem.OEThrow.Fatal("Unable to open " + ifname + " for reading");
        }

        oemolostream ofs = null;
        if (itf.HasString("-out"))
        {
            string ofname = itf.GetString("-out");
            ofs = new oemolostream();
            if (!ofs.open(ofname))
                OEChem.OEThrow.Fatal("Unable to open " + ofname + " for writing");
            if (!overlay)
```

(continues on next page)

(continued from previous page)

```
                OEChem.OEThrow.Warning("Output is the same as input when overlay is false
↪");
        }

        foreach (OEMCMolBase mol in ifs.GetMCMolBases())
        {
            OEChem.OEThrow.Info(mol.GetTitle());

            uint maxIdx = mol.GetMaxConfIdx();
            double[] rmsds = new double[maxIdx];
            double[] rmtx = new double[9 * maxIdx];
            double[] tmtx = new double[3 * maxIdx];

            // performing RMSD for all conformers
            OEChem.OERMSD(rmol, mol, rmsds, automorph, heavy, overlay, rmtx, tmtx);

            foreach (OEConfBase conf in mol.GetConfs())
            {
                uint cidx = conf.GetIdx();
                OEChem.OEThrow.Info("Conformer " + cidx + ": rmsd = " + rmsds[cidx]);

                if (overlay)
                {
                    OEChem.OERotate(conf, RMSD.slice(rmtx, cidx * 9, cidx * 9 + 9));
                    OEChem.OETranslate(conf, RMSD.slice(tmtx, cidx * 3, cidx * 3 + 3));
                }
            }

            if (ofs != null)
            {
                OEChem.OEWriteMolecule(ofs, mol);
            }
        }

        if (ofs != null)
        {
            ofs.close();
        }
    }

    private static double[] slice(double[] src, uint begin, uint end)
    {
        double[] ret = new double[end - begin];
        uint cnt = 0;
        for (uint i = begin; i != end; i++, cnt++)
        {
            ret[cnt] = src[i];
        }
        return ret;
    }

    public static void Main(String[] args)
```

(continues on next page)

(continued from previous page)

```
    {
        OEInterface itf = new OEInterface();
        OEChem.OEConfigure(itf, InterfaceData);
        if (OEChem.OEParseCommandLine(itf, args, "RMSD"))
            RMSDUtility(itf);
    }

    private static String InterfaceData = @"
!BRIEF [options] [-ref <mol file>] [-in <mol file>] [-out <mol file>]

!CATEGORY ""input/output options""

  !PARAMETER -ref
    !TYPE string
    !REQUIRED true
    !BRIEF input reference mol file name
    !KEYLESS 1
  !END

  !PARAMETER -in
    !ALIAS -i
    !TYPE string
    !REQUIRED true
    !BRIEF input mol file name
    !KEYLESS 2
  !END

  !PARAMETER -out
    !ALIAS -o
    !TYPE string
    !REQUIRED false
    !BRIEF output file name, this implies that -overlay should be true
    !KEYLESS 3
  !END

!END

!CATEGORY ""options""

  !PARAMETER -automorph
    !TYPE bool
    !DEFAULT true
    !BRIEF assign best atom association
    !DETAIL
        If false, atoms are associated by order.
        If true, graph isomorphism is determined with symmetry perception.
  !END

  !PARAMETER -overlay
    !TYPE bool
    !DEFAULT true
    !BRIEF Minimize to the smallest RMSD
```

(continues on next page)

(continued from previous page)

```
  !END

  !PARAMETER -heavyonly
    !TYPE bool
    !DEFAULT true
    !BRIEF Ignore hydrogens for RMSD calculation
  !END

  !PARAMETER -verbose
    !ALIAS -v
    !TYPE bool
    !DEFAULT false
    !BRIEF verbose
  !END

!END
";
}
```

See also

- *OERMSD* function
- *OERotate* function
- *OETranslate* function

Modifying SD tags

A program that can modify the SD tags on an SD molecule. The flags are: `-remove`, a list of property tags to be removed (non-matching tags are kept), `-keep`, a list of property tags to be kept (non-matching tags are removed), `-clearAll`, all SD tags are removed. The program formats are limited to those that can have SD data `i.e.` SDF, OEB, and CSV.

Example

```
prompt > SDFModProps -keep NAME ID_NUM -i dbase.sdf -o output.sdf
prompt > SDFModProps -clearAll -i dbase.sdf -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
```

(continues on next page)

(continued from previous page)

```
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Modifies the SD data of a set of input molecules by clearing all tags,
* defining which tags to keep or defining which tags to remove
*****************************************************************************/
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class SDFModProps
{
    public static void ClearProps(oemolistream ifs,
                                  oemolostream ofs)
    {
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEClearSDData(mol);
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    public static void KeepProps(HashSet<string> props,
                                 oemolistream ifs,
                                 oemolostream ofs)
    {
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            foreach (OESDDataPair dp in OEChem.OEGetSDDataPairs(mol))
            {
                if (!props.Contains(dp.GetTag()))
                {
                    OEChem.OEDeleteSDData(mol, dp.GetTag());
                }
            }
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    public static void RemoveProps(HashSet<string> props,
                                   oemolistream ifs,
                                   oemolostream ofs)
    {
        OEGraphMol mol = new OEGraphMol();
```

(continues on next page)

(continued from previous page)

```
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            foreach (string prop in props)
            {
                OEChem.OEDeleteSDData(mol, prop);
            }
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    private static HashSet<string> FillSet(OEStringIter proplist)
    {
        HashSet<string> props = new HashSet<string>();
        foreach (string prop in proplist)
        {
            props.Add(prop);
        }
        return props;
    }

    public static void ModProps(OEInterface itf,
                                oemolistream ifs,
                                oemolostream ofs)
    {
        if (itf.HasString("-keep"))
        {
            OEStringIter proplist = itf.GetStringList("-keep");
            HashSet<string> props = FillSet(proplist);
            KeepProps(props, ifs, ofs);
        }
        else if (itf.HasString("-remove"))
        {
            OEStringIter proplist = itf.GetStringList("-remove");
            HashSet<string> props = FillSet(proplist);
            RemoveProps(props, ifs, ofs);
        }
        else if (itf.GetBool("-clearAll"))
        {
            ClearProps(ifs, ofs);
        }
    }

    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "SDFModProps", args);

        int numoption = 0;
        if (itf.HasString("-keep"))
        {
            numoption++;
        }
        if (itf.HasString("-remove"))
```

(continues on next page)

(continued from previous page)

```
        {
            numoption++;
        }
        if (itf.GetBool("-clearAll"))
        {
            numoption++;
        }
        if (numoption != 1)
        {
            OEChem.OEThrow.Usage("Need to pick one from -keep, -remove, or -clearAll");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }
        if (!OEChem.OEIsSDDataFormat(ifs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Only works for input file formats that support SD data␣
↪(sdf,oeb,csv)");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(itf.GetString("-o")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for writing
↪");
        }
        if (!OEChem.OEIsSDDataFormat(ofs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Only works for output file formats that support SD␣
↪data (sdf,oeb,csv)");
        }

        ModProps(itf, ifs, ofs);
    }

    private static string interfaceData=@"
!BRIEF [-remove] <prop1 prop2...> [-keep] <prop1 prop2...> [-clearAll] -i <input> -o
↪<output>
!PARAMETER -i
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
!END
!PARAMETER -o
  !ALIAS -out
  !TYPE string
  !REQUIRED true
```

(continues on next page)

(continued from previous page)

```
  !BRIEF Output file name
!END
!PARAMETER -keep
  !ALIAS -k
  !TYPE string
  !LIST true
  !BRIEF SD tags to be kept
!END
!PARAMETER -remove
  !ALIAS -r
  !TYPE string
  !LIST true
  !BRIEF SD tags to be removed
!END
!PARAMETER -clearAll
  !ALIAS -c
  !TYPE bool
  !DEFAULT false
  !BRIEF Removes all SD tags
!END
";
}
```

See also

- *OEDeleteSDData* function
- *OEClearSDData* function

Exporting SD data to a CSV file

A program that converts the tags on an SD file to CSV format. The output has on the first line the molecule title, followed by all the unique tags found in the input database.

Example

```
prompt > SDF2CSV input.sdf output.csv
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
```

(continues on next page)

(continued from previous page)

```
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Extract properties from SD file and save as CSV
*****************************************************************************/
using System;
using System.Collections.Generic;
using System.IO;
using OpenEye.OEChem;

public class SDF2CSV
{
    public static void SDFtoCSV(oemolistream ifs, StreamWriter csv)
    {
        List<string> tagList = new List<string>();
        OEGraphMol mol = new OEGraphMol();
        // read through once to find all unique tags
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            foreach (OESDDataPair dp in OEChem.OEGetSDDataPairs(mol))
            {
                if (!tagList.Contains(dp.GetTag()))
                {
                    tagList.Add(dp.GetTag());
                }
            }
        }

        // output the header row
        csv.Write("Title");
        foreach (string tag in tagList)
        {
            csv.Write(",{0}", tag);
        }
        csv.WriteLine();

        // read through again filling rows for each molecule
        ifs.rewind();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            csv.Write("{0}", mol.GetTitle());
            foreach (string tag in tagList)
            {
                csv.Write(",{0}", OEChem.OEGetSDData(mol, tag));
            }
            csv.WriteLine();
```

(continues on next page)

(continued from previous page)

```
        }
    }

    public static void Main(string[] args)
    {
        if (args.Length != 2)
        {
            OEChem.OEThrow.Usage("SDF2CSV <infile> <csvfile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }
        uint fmt = ifs.GetFormat();
        if (fmt != OEFormat.SDF && fmt != OEFormat.OEB)
        {
            OEChem.OEThrow.Fatal("Only works for sdf or oeb input files");
        }

        StreamWriter csv = null;
        try
        {
            csv = new StreamWriter(args[1]);
            SDFtoCSV(ifs, csv);
            csv.Close();
        }
        catch
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
        }
    }
}
```

See also

- *OEGetSDData* function
- *OEGetSDDataPairs* function

Adding csv data as SD tags

A program that adds the data in a csv file as SD tags to a database of molecules. The csv data and molecules are paired by the molecule title.

Example

```
prompt > CSV2SDF input.csv output.sdf
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Merge a CSV file of data/properties, key on compound name in first column
* and use column titles as keys.  All data is read/written as strings
*****************************************************************************/
using System;
using System.IO;
using System.Collections.Generic;
using System.Collections;
using OpenEye.OEChem;

public class CSV2SDF
{
    public static void CSVtoSDF(StreamReader csv,
                                oemolistream ifs,
                                oemolostream ofs)
    {
        string line = csv.ReadLine();

        ArrayList propnameslist = new ArrayList(line.Split(new char[] { ',' }));
        propnameslist.RemoveAt(0);

        Dictionary<string, ArrayList> values = new Dictionary<string, ArrayList>();
        while ( (line = csv.ReadLine()) != null )
        {
            ArrayList tokenlist = new ArrayList(line.Split(new char[] { ',' }));
            if (tokenlist.Count > 1)
            {
                string title = (string)tokenlist[0];
                if (string.IsNullOrEmpty(title))
                {
                    OEChem.OEThrow.Warning("Skipping entry with no title");
                    continue;
                }
```

(continues on next page)

(continued from previous page)

```
                tokenlist.RemoveAt(0);
                values.Add(title, tokenlist);
            }
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (values.ContainsKey(mol.GetTitle()))
            {
                ArrayList vallist = new ArrayList();
                values.TryGetValue(mol.GetTitle(), out vallist);
                for (int p = 0; p < vallist.Count && p < propnameslist.Count; ++p)
                {
                    string sdval = (string)vallist[p];
                    if (string.IsNullOrEmpty(sdval))
                    {
                        continue;
                    }
                    else
                    {
                        OEChem.OESetSDData(mol, (string)propnameslist[p], sdval);
                    }
                }
            }
            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    public static void Main(string[] args)
    {
        if (args.Length != 3)
        {
            OEChem.OEThrow.Usage("CSV2SDF <csvfile> <infile> <outsdfile>");
        }

        StreamReader csv = null;
        try
        {
            csv = new System.IO.StreamReader(args[0]);
        }
        catch
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }
        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for reading");
        }

        oemolostream ofs = new oemolostream();
```

(continues on next page)

(continued from previous page)

```
        if (!ofs.open(args[2]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[2] + " for writing");
        }
        uint fmt = ofs.GetFormat();
        if (fmt != OEFormat.SDF && fmt != OEFormat.OEB)
        {
            OEChem.OEThrow.Fatal("Only works for sdf or oeb output files");
        }

        CSVtoSDF(csv, ifs, ofs);
    }
}
```

See also

- *OESetSDData* function
- *OEStringTokenize* function

Renaming molecules by SD field

A program that renames a molecule by a particular SD tag attached to that molecule.

Example

```
prompt > SDFRename ID_NUM input.sdf output.smi
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Rename SDF molecules by specified field
```

(continues on next page)

(continued from previous page)

```
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class SDFRename
{
    public static void ProcessFile(string field, oemolistream ifs, oemolostream ofs)
    {
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (OEChem.OEHasSDData(mol, field))
            {
                mol.SetTitle(OEChem.OEGetSDData(mol, field));
            }
            else
            {
                OEChem.OEThrow.Warning("Renaming of molecule " + mol.GetTitle() + "␣
↪failed; no field " + field);
            }

            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    public static void Main(string[] args)
    {
        if (args.Length != 3)
        {
            OEChem.OEThrow.Usage("SDFRename <fieldname> <infile> <outfile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for reading");
        }

        if (!OEChem.OEIsSDDataFormat(ifs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Only works for input file formats that support SD data␣
↪(sdf,oeb,csv)");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[2]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[2] + " for writing");
        }

        ProcessFile(args[0], ifs, ofs);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

See also

- *OEHasSDData* function
- *OEMolBase.SetTitle* method

Filter molecules by SD data

A program that can filter a database of molecules based on SD tags. Any SD tag that contains numerical data can be used. Flags are: `-tag`, the SD tag to use, `-min`, the minimum value of the tag, `-max`, the maximum value of the tag. The program formats are limited to those that can have SD data `i.e.` SDF, OEB, and CSV.

Example

```
prompt > SDFilter -tag LOGD -min -2 -max 5 -i input.sdf -o output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Filter molecules by SD data
//****************************************************************************/
using System;
using OpenEye.OEChem;

public class SDFilter
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "SDFilter", args);
        if (!(itf.HasDouble("-min") || itf.HasDouble("-max")))
```

(continues on next page)

(continued from previous page)

```
        {
            OEChem.OEThrow.Fatal("Please set a filter value with -min or -max");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }
        if (!OEChem.OEIsSDDataFormat(ifs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Only works for input file formats that support SD data␣
↪(sdf,oeb,csv)");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(itf.GetString("-o")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for writing
↪");
        }
        if (!OEChem.OEIsSDDataFormat(ofs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Only works for output file formats that support SD␣
↪data (sdf,oeb,csv)");
        }

        String tag = itf.GetString("-tag");

        double minval = double.MinValue;
        if (itf.HasDouble("-min"))
        {
            minval = itf.GetDouble("-min");
        }

        double maxval = double.MaxValue;
        if (itf.HasDouble("-max"))
        {
            maxval = itf.GetDouble("-max");
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            if (!OEChem.OEHasSDData(mol, tag))
            {
                OEChem.OEThrow.Warning("Unable to find " + tag + " on " + mol.
↪GetTitle());
                continue;
            }
            double tagvalue;
```

(continues on next page)

(continued from previous page)

```
            string value = OEChem.OEGetSDData(mol, tag);
            if (!double.TryParse(value.Trim(), out tagvalue))
            {
                OEChem.OEThrow.Warning("Failed to convert (" + value + ") to a number in
↪" + mol.GetTitle());
                continue;
            }
            if (tagvalue < minval)
            {
                continue;
            }
            if (tagvalue > maxval)
            {
                continue;
            }

            OEChem.OEWriteMolecule(ofs, mol);
        }
    }

    private static string interfaceData=@"
!BRIEF -i <input> -o <output> -tag <name> [-min <num>] [-max <num>]
!PARAMETER -i
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !TYPE string
  !REQUIRED true
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -tag
  !TYPE string
  !REQUIRED true
  !BRIEF SD tag
!END
!PARAMETER -min
  !TYPE double
  !REQUIRED false
  !BRIEF minimum value of SD tag
!END
!PARAMETER -max
  !TYPE double
  !REQUIRED false
  !BRIEF maximum value of SD tag
!END
";
}
```

See also

- *OEHasSDData* function
- *OEGetSDData* function
- *OEStringToNumber* function

Print OEChem and file format information

A program that prints the current OEChem version, platform, build date, and the release name of the overall toolkits being used. It also prints details of all the molecule file formats supported by the OEChem toolkit and whether they are readable and/or writable.

Example

```
prompt> OEChemInfo
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Print toolkit release date platform and build information. Also print out
* all formats supported by OEChem and whether they are readable or writable
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class OEChemInfo
{
    public static void PrintFormats()
    {
        Console.WriteLine("\tcode| ext           | description                          ␣
↪   |read? |write?");
        Console.WriteLine("\t----+---------------+--------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪---+------+------");
        for (uint code = 1; code < OEFormat.MAXFORMAT; ++code)
        {
            string extension = OEChem.OEGetFormatExtension(code);
            string description = OEChem.OEGetFormatString(code);
            string readable = OEChem.OEIsReadable(code) ? "yes" : "no";
            string writeable = OEChem.OEIsWriteable(code) ? "yes" : "no";
            Console.WriteLine("\t {0,2} | {1,-13} | {2,-40} | {3,-4} | {4,-4}",
                              code, extension, description, readable, writeable);
        }
        Console.WriteLine("\t----+---------------+----------------------------------------
↪---+------+------");
    }

    public static void Main()
    {
        Console.WriteLine("Installed OEChem version: {0} platform: {1} built: {2}␣
↪release name: {3}",
                          OEChem.OEChemGetRelease(),
                          OEChem.OEChemGetPlatform(),
                          OEChem.OEChemGetVersion(),
                          OEChem.OEToolkitsGetRelease());
        PrintFormats();
    }
}
```

See also

- *OEGetFormatString* function
- *OEGetFormatExtension* function
- *OEChemGetRelease* function
- *OEChemGetVersion* function
- *OEChemGetPlatform* function
- *OEToolkitsGetRelease* function

Counting molecules

A program that counts the number of molecules, or the number of conformers, in an input molecule file, or files. The flag `-conf` is used to count conformers. The output is the total molecules or conformers per file, and the total sum if more than one file is used as input. If conformers are counted the average conformer count per molecule is also output.

Example

```
prompt> MolCount -conf input1.oeb.gz input2.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Counts molecule (and conformers) in input files
//****************************************************************************/
using System;
using OpenEye.OEChem;

public class MolCount
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "MolCount", args);

        bool confomega = itf.GetBool("-confomega");
        bool confs = (confomega || itf.GetBool("-conf"));

        long total = 0;
        long totconfs = 0;
        long nfiles = 0;

        foreach (string filename in itf.GetStringList("-i"))
        {
            oemolistream ifs = new oemolistream();
            if (!ifs.open(filename))
            {
                OEChem.OEThrow.Warning("Unable to open " + filename + " for reading");
                continue;
            }
            if (confomega)
```

(continues on next page)

(continued from previous page)

```
            {
                ifs.SetConfTest(new OEOmegaConfTest(false));
            }

            nfiles++;
            long count = 0;
            long nconfs = 0;

            OEMol mol = new OEMol();
            while (OEChem.OEReadMolecule(ifs, mol))
            {
                count++;
                if (confs)
                {
                    nconfs += mol.NumConfs();
                }
            }
            Console.WriteLine(filename + " contains " + count + " molecule(s).");
            if (confs)
            {
                Console.WriteLine("Total # of conformers:   " + nconfs);
                Console.WriteLine("Average # of conformers: " + (float)nconfs /␣
↪(float)count);
                Console.WriteLine("------------------------------------------------------------
↪-----");
            }
            total += count;
            totconfs += nconfs;
        }
        Console.WriteLine("============================================================");
        Console.WriteLine("Total " + total + " molecules");
        if (confs && (nfiles > 0) )
        {
            Console.WriteLine("Total # of conformers:   " + totconfs);
            Console.WriteLine("Average # of conformers: " + (float)totconfs /␣
↪(float)total);
        }
    }

    private static string interfaceData=@"
!BRIEF [-conf] [-i] <infile1> [<infile2>...]
!PARAMETER -i
  !ALIAS -in
  !TYPE string
  !LIST true
  !REQUIRED true
  !BRIEF Input file name(s)
  !KEYLESS 1
!END
!PARAMETER -conf
  !ALIAS -c
  !TYPE bool
```

(continues on next page)

(continued from previous page)

```
  !DEFAULT false
  !BRIEF Count conformers
!END
!PARAMETER -confomega
  !TYPE bool
  !DEFAULT false
  !BRIEF Count Omega conformers
!END
";
}
```

Get molecule titles

A program that prints the molecule titles found in an input molecule database. The program will output the titles to stdout if no output file is set. Molecules with no title are shown as "untitled".

Example

```
prompt> GetTitles input.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Output all molecule titles
//****************************************************************************/
using System;
using System.IO;
using OpenEye.OEChem;

public class GetTitles
{
    public static void Main(string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        if (args.Length < 1 || args.Length >2)
        {
            OEChem.OEThrow.Usage("GetTitles <infile> [<outfile>]");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        StreamWriter sw = null;
        if (args.Length > 1)
        {
            try
            {
                sw = new StreamWriter(args[1]);
                Console.SetOut(sw);
            }
            catch
            {
                OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
            }
        }

        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            string title = mol.GetTitle();
            if (string.IsNullOrEmpty(title))
            {
                title = "untitled";
            }
            Console.WriteLine(title);
        }
        if (args.Length > 1)
        {
            sw.Close();
        }
    }
}
```

See also

- *OEMolBase.GetTitle* method

Print simple molecule information

A program that prints some simple molecule information for a database of molecules. This includes the title, molecular weight, number of atoms, number of heavy atoms, number of ring atoms, number of rotors and number of conformers.

Example

```
prompt> MolStats input.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Output some basic molecule properties
//****************************************************************************/
using System;
using OpenEye.OEChem;

public class MolStats
{
    public static void Main(string[] args)
    {
        if (args.Length != 1)
        {
            OEChem.OEThrow.Usage("MolStats <infile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
```

(continues on next page)

(continued from previous page)

```
        }

        Console.WriteLine("Title MolWt NumAtoms NumHeavyAtoms NumRingAtoms NumRotors␣
↪NumConfs");
        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            string title = mol.GetTitle();
            if (string.IsNullOrEmpty(title))
            {
                title = "Untitled";
            }
            Console.WriteLine(title + " {0:F3} {1} {2} {3} {4} {5}",
                              OEChem.OECalculateMolecularWeight(mol),
                              mol.NumAtoms(),
                              OEChem.OECount(mol, new OEIsHeavy()),
                              OEChem.OECount(mol, new OEAtomIsInRing()),
                              OEChem.OECount(mol, new OEIsRotor()),
                              mol.NumConfs());
        }
    }
}
```

See also

- *OEIsHeavy* functor
- *OEIsRotor* functor
- *OECalculateMolecularWeight* function

Print rotor count distribution

A program that prints the rotorcount distribution for a database of molecules. It uses *OEIsRotor* functor to determine if a bond is freely rotatable. Use the `OEGetRotatableBondCount` function to get a more precise count of rotatable bonds in a molecule.

Example

```
prompt > RotorCount input.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
```

(continues on next page)

(continued from previous page)

```
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Counts the number of rotatable bonds in the input compound file and
* outputs the maximum number of rotomers and the rotomer distribution
*****************************************************************************/
using System;
using System.Collections.Generic;
using OpenEye.OEChem;

public class RotorCount
{
    public static void Main(string[] args)
    {
        if (args.Length != 1)
        {
            OEChem.OEThrow.Usage("RotorCount <infile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        List<int> rotcounts = new List<int>();
        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            uint nrots = OEChem.OECount(mol, new OEIsRotor());
            while (nrots >= rotcounts.Count)
            {
                rotcounts.Add(0);
            }
            rotcounts[(int)nrots] += 1;
        }

        int maxrots = rotcounts.Count-1;
        Console.WriteLine("Max rotors: " + maxrots);
        Console.WriteLine("Rotorcount distribution:");
        for (int i = 0; i < rotcounts.Count; i++)
        {
            Console.WriteLine("\t" + i + ":\t" + rotcounts[i]);
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

See also

- *OEIsRotor* functor
- `OEGetRotatableBondCount` function

Find minimum path in a molecule

A program that finds and outputs the minimum path in a molecule. The input can be two specific named atoms or two SMARTS matches. As SMARTS matches could hit more than one atom this could result in multiple different paths. To use atom names a helper program, printatomnames, is provided (see below). If `-o` is set then the atoms in the path will be output to a molecule file, otherwise just the length of the path is reported. `-verbose` outputs more details on the path.

Example

```
prompt> MinPath -i input.oeb.gz -atom1 'N1' -atom2 'S1'
prompt> MinPath -i input.oeb.gz -smarts1 'c1cccs1' -smarts2 'OC(=O)C' -o output.smi
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Find the minimum path length between 2 smarts patterns
* or the path length between 2 named atoms
*****************************************************************************/
using System;
using System.Collections.Generic;
using OpenEye.OEChem;
```

(continues on next page)

(continued from previous page)

```
public class MinPath
{
    public class Pair<T, U>
    {
        public Pair()
        {
        }

        public Pair(T first, U second)
        {
            this.First = first;
            this.Second = second;
        }

        public T First
        {
            get;
            set;
        }
        public U Second
        {
            get;
            set;
        }
    }

    public static void AtomPathLength(oemolistream ifs, oemolostream ofs,
                                      OEInterface itf,
                                      String atm1, String atm2)
    {
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OETriposAtomNames(mol);
            OEAtomBase a1 = null;
            OEAtomBase a2 = null;
            foreach (OEAtomBase atm in mol.GetAtoms())
            {
                if (atm1.Equals(atm.GetName()))
                {
                    a1 = atm;
                }
                if (atm2.Equals(atm.GetName()))
                {
                    a2 = atm;
                }
                if ((object)a1 != null && (object)a2 != null)
                {
                    break;
                }
            }
```

(continues on next page)

(continued from previous page)

```
            if ((object)a1 == null || (object)a2 == null)
            {
                OEChem.OEThrow.Warning("Failed to find atoms " + atm1 + " and " + atm2 +
↪" in molecule");
                continue;
            }

            uint pathlen = OEChem.OEGetPathLength(a1, a2);
            if (itf.GetBool("-verbose") || !itf.HasString("-o"))
            {
                String smiles = OEChem.OECreateIsoSmiString(mol);
                Console.WriteLine("Path length: " + pathlen + " in " + smiles);
            }

            OEAtomBaseIter spath = OEChem.OEShortestPath(a1, a2);
            OEGraphMol spathmol = new OEGraphMol();
            bool adjustHCount = true;
            OEChem.OESubsetMol(spathmol, mol, new OEIsAtomMember(spath), adjustHCount);
            String spathsmiles = OEChem.OECreateIsoSmiString(spathmol);

            if (itf.HasString("-o"))
            {
                OEChem.OEWriteMolecule(ofs, spathmol);
            }
            else if (itf.GetBool("-verbose"))
            {
                Console.WriteLine(spathsmiles);
            }
        }
    }

    public static void SmartsPathLength(oemolistream ifs, oemolostream ofs,
                                        OEInterface itf,
                                        OESubSearch ss1, OESubSearch ss2)
    {
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEPrepareSearch(mol, ss1);
            OEChem.OEPrepareSearch(mol, ss2);
            if (!(ss1.SingleMatch(mol) && ss2.SingleMatch(mol)))
            {
                OEChem.OEThrow.Warning("Unable to find SMARTS matches in "
                                       + mol.GetTitle() + ", skipping");
                continue;
            }

            List<Pair<OEAtomBase, OEAtomBase> > allatompairs = new List<Pair<OEAtomBase,␣
↪OEAtomBase> >();
            bool unique = true;
            int allminlen = int.MaxValue;
            foreach (OEMatchBase match1 in ss1.Match(mol, unique))
```

(continues on next page)

(continued from previous page)

```
            {
                foreach (OEMatchBase match2 in ss2.Match(mol, unique))
                {
                    List<Pair<OEAtomBase, OEAtomBase> > atompairs = new List<Pair
↪<OEAtomBase, OEAtomBase> >();
                    int minlen = int.MaxValue;
                    foreach (OEMatchPairAtom mp1 in match1.GetAtoms())
                    {
                        foreach (OEMatchPairAtom mp2 in match2.GetAtoms())
                        {
                            uint pathlen = OEChem.OEGetPathLength(mp1.target, mp2.
↪target);
                            if (minlen > pathlen)
                            {
                                minlen = (int)pathlen;
                                atompairs.Clear();
                                atompairs.Add(new Pair<OEAtomBase, OEAtomBase>(mp1.
↪target, mp2.target));
                            }
                            else if (minlen == pathlen)
                            {
                                atompairs.Add(new Pair<OEAtomBase, OEAtomBase>(mp1.
↪target, mp2.target));
                            }
                        }
                    }
                    if (minlen < allminlen)
                    {
                        allminlen = minlen;
                        allatompairs = atompairs;
                    }
                    else if (minlen == allminlen)
                    {
                        allatompairs.AddRange(atompairs);
                    }
                }
            }

            if (itf.GetBool("-verbose") || !itf.HasString("-o"))
            {
                Console.WriteLine("Shortest path length: " + allminlen + " in " + OEChem.
↪OECreateIsoSmiString(mol));
            }

            HashSet<String> spathlist = new HashSet<String>();
            foreach (Pair<OEAtomBase, OEAtomBase> pair in allatompairs)
            {
                OEAtomBaseIter spath = OEChem.OEShortestPath((OEAtomBase)pair.First,␣
↪(OEAtomBase)pair.Second);
                OEGraphMol spathmol = new OEGraphMol();
                OEChem.OESubsetMol(spathmol, mol, new OEIsAtomMember(spath));
                String spathsmiles = OEChem.OECreateIsoSmiString(spathmol);
```

(continues on next page)

(continued from previous page)

```
                if (!spathlist.Add(spathsmiles))
                {
                    continue;
                }
                if (itf.HasString("-o"))
                {
                    OEChem.OEWriteMolecule(ofs, spathmol);
                }
                else if (itf.GetBool("-verbose"))
                {
                    Console.WriteLine(spathsmiles);
                }
            }
        }
    }

    public static void Main(String[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "MinPath", args);

        if (!((itf.HasString("-smarts1") && itf.HasString("-smarts2"))
                ^ (itf.HasString("-atom1") && itf.HasString("-atom2"))))
        {
            OEChem.OEThrow.Fatal("-smarts1 and -smarts2 or -atom1 and -atom2 must be set
↪");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(itf.GetString("-i")))
        {
            OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-i") + " for reading
↪");
        }

        oemolostream ofs = new oemolostream();
        if (itf.HasString("-o"))
        {
            if (!ofs.open(itf.GetString("-o")))
            {
                OEChem.OEThrow.Fatal("Unable to open " + itf.GetString("-o") + " for␣
↪writing");
            }
        }

        if (itf.HasString("-smarts1") && itf.HasString("-smarts2"))
        {
            OESubSearch ss1 = new OESubSearch();
            String smarts1 = itf.GetString("-smarts1");
            if (!ss1.Init(smarts1))
            {
                OEChem.OEThrow.Fatal("Unable to parse SMARTS1: " + smarts1);
            }
```

(continues on next page)

(continued from previous page)

```
            OESubSearch ss2 = new OESubSearch();
            String smarts2 = itf.GetString("-smarts2");
            if (!ss2.Init(smarts2))
            {
                OEChem.OEThrow.Fatal("Unable to parse SMARTS2: " + smarts2);
            }

            SmartsPathLength(ifs, ofs, itf, ss1, ss2);
            ofs.close();
        }
        else
        {
            String atom1 = itf.GetString("-atom1");
            String atom2 = itf.GetString("-atom2");
            AtomPathLength(ifs, ofs, itf, atom1, atom2);
        }
    }
    private static string interfaceData=@"
!BRIEF -i <input> [-o <output>] -smarts1 <smarts> -smarts2 <smarts> | -atom1 <name> -
↪atom2 <name>
!PARAMETER -i
  !TYPE string
  !REQUIRED true
  !BRIEF Input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !TYPE string
  !REQUIRED false
  !BRIEF Output file name
  !KEYLESS 2
!END
!PARAMETER -smarts1
  !TYPE string
  !BRIEF Smarts pattern to identify 1st atom
!END
!PARAMETER -smarts2
  !TYPE string
  !BRIEF Smarts pattern to identify 2nd atom
!END
!PARAMETER -atom1
  !TYPE string
  !BRIEF Name of 1st atom
!END
!PARAMETER -atom2
  !TYPE string
  !BRIEF Name of 2nd atom
!END
!PARAMETER -verbose
  !TYPE bool
  !REQUIRED false
```

(continues on next page)

(continued from previous page)

```
  !DEFAULT false
  !BRIEF Print verbose output
!END
";
}
```

See also

- *OEIsMemberPtr* specialized functor *OEIsAtomMember*
- *OEGetPathLength* function
- *OEShortestPath* function
- *OEPrepareSearch* function

Print molecule atom names

A program that prints the Tripos atom names for each atom in a molecule. Useful for specifying atoms in the minpath example (see above).

Example

```
prompt> PrintAtomNames input.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Print out atom names
//****************************************************************************/
using System;
using OpenEye.OEChem;
```

(continues on next page)

(continued from previous page)

```
public class PrintAtomNames
{

    public static void Main(string[] args)
    {
        if (args.Length != 1)
        {
            OEChem.OEThrow.Usage("PrintAtomNames <infile>");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OETriposAtomNames(mol);
            foreach (OEAtomBase atom in mol.GetAtoms())
            {
                Console.WriteLine(atom.GetName());
            }
        }
    }
}
```

See also

- *OETriposAtomNames* function

Extract 2D ring templates

A program that extracts anonymized ring systems that can be potential new ring templates in the **OEChem TK** 2D coordinate generation system.

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
```

(continues on next page)

(continued from previous page)

```
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Extract ring templates for 2D coordinate generation
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class ExtractRingTemplates
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "ExtractRingTemplates", args);

        String ifname = itf.GetString("-in");
        String ofname = itf.GetString("-out");

        oemolistream ifs = new oemolistream();
        if (!ifs.open(ifname))
            OEChem.OEThrow.Fatal("Unable to open " + ifname + " for reading");

        if (!OEChem.OEIs2DFormat(ifs.GetFormat()))
            OEChem.OEThrow.Fatal("Invalid input format: need 2D coordinates");

        oemolostream ofs = new oemolostream();
        if (!ofs.open(ofname))
            OEChem.OEThrow.Fatal("Unable to open " + ofname + " for writing");

        if (!OEChem.OEIs2DFormat(ofs.GetFormat()))
            OEChem.OEThrow.Fatal("Invalid output format: unable to write 2D coordinates
↪");

        uint nrrings = 0;
        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
             foreach (OEMolBase ring in OEChem.OEExtractRingTemplates(mol))
             {
                  nrrings += 1;
                  OEChem.OEWriteMolecule(ofs, ring);
             }
        }
        OEChem.OEThrow.Info(nrrings + " number of ring templates extracted");
    }

    private static string interfaceData=@"
!BRIEF [-i] <input> [-o] <output>
!PARAMETER -i
```

(continues on next page)

(continued from previous page)

```
  !ALIAS -in
  !TYPE string
  !REQUIRED true
  !BRIEF input file name
  !KEYLESS 1
!END
!PARAMETER -o
  !ALIAS -out
  !TYPE string
  !REQUIRED true
  !BRIEF output file name
  !KEYLESS 2
!END
";
}
```

See also

- *OEExtractRingTemplateOptions* class
- *OEExtractRingTemplates* function

Create 2D ring dictionary

A program that creates a dictionary of 2D ring layouts that can be plugged into the 2D coordinates generation system.

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
 * Creates a new 2D ring dictionary
 ****************************************************************************/

using System;
```

(continues on next page)

(continued from previous page)

```
using OpenEye.OEChem;

public class CreateRingDict
{
    public static int Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "CreateRingDict", args);

        string ifname = itf.GetString("-in");
        string ofname = itf.GetString("-ringdict");

        oemolistream ifs = new oemolistream();
        if (!ifs.open(ifname))
        {
            OEChem.OEThrow.Fatal("Cannot open input file for reading!");
        }

        if (!OEChem.OEIs2DFormat(ifs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input file format for 2D coordinates!");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(ofname))
        {
            OEChem.OEThrow.Fatal("Unable to open output file for writing!");
        }

        if (ofs.GetFormat() != OEFormat.OEB)
        {
            OEChem.OEThrow.Fatal("Output file has to have OEB format!");
        }

        OE2DRingDictionaryCreatorOptions opts = new OE2DRingDictionaryCreatorOptions();
        opts.SetRetainExistingBuiltInTemplates(itf.GetBool("-retain-built-in"));
        OE2DRingDictionary ringdict = new OE2DRingDictionary(opts);

        OEDots dots = new OEDots(10000, 100, "molecules");

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            dots.Update();
            ringdict.AddRings(mol);
        }
        dots.Total();

        uint nrrings = ringdict.NumRings();
        OEChem.OEThrow.Info(nrrings.ToString() + " number of ring templates have been␣
↪extracted!");

        if (nrrings != 0)
```

(continues on next page)

(continued from previous page)

```
        {
            OEChem.OEWrite2DRingDictionary(ofname, ringdict);
        }

        return 0;
    }
private static string interfaceData = @"
!BRIEF [-in] <input> [-ringdict] <output ringdict>

!CATEGORY ""input/output options :""

  !PARAMETER -in
    !ALIAS -i
    !TYPE string
    !REQUIRED true
    !KEYLESS 1
    !VISIBILITY simple
    !BRIEF Input 2D molecule filename
  !END

  !PARAMETER -ringdict
    !ALIAS -rd
    !TYPE string
    !REQUIRED true
    !KEYLESS 2
    !VISIBILITY simple
    !BRIEF Output ring dictionary OEB filename
  !END

!END

!CATEGORY ""ring dictionary options :""

  !PARAMETER -retain-built-in
    !TYPE bool
    !REQUIRED false
    !DEFAULT false
    !VISIBILITY simple
    !BRIEF Ignore ring template if built-in exists
  !END

!END
";
}
```

See also

- *OE2DRingDictionary* class
- *OE2DRingDictionaryCreatorOptions* class
- *OEWrite2DRingDictionary* function

Append 2D ring dictionary

A program that adds new ring layouts to a ring dictionary that can be plugged into the 2D coordinates generation system.

Note

If a ring template already exists in the dictionary then the new template will be ignored *i.e.* currently ring templates can not be overwritten in the *OE2DRingDictionary* object.

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
 * Appends rings to an existing 2D rings dictionary
 ****************************************************************************/
using System;
using OpenEye.OEChem;

public class AppendRingDict
{
    public static int Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "AppendRingDict", args);

        string ifname = itf.GetString("-in");
        string irdfname = itf.GetString("-inringdict");
        string ordfname = itf.GetString("-outringdict");
```

(continues on next page)

(continued from previous page)

```
        oemolistream ifs = new oemolistream();
        if (!ifs.open(ifname))
        {
            OEChem.OEThrow.Fatal("Cannot open input file for reading!");
        }

        if (!OEChem.OEIs2DFormat(ifs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid input file format for 2D coordinates!");
        }

        if (!OEChem.OEIsValid2DRingDictionary(irdfname))
        {
            OEChem.OEThrow.Fatal("Invalid ring dictionary file!");
        }

        OE2DRingDictionary ringdict = new OE2DRingDictionary(irdfname);

        uint nrrings = ringdict.NumRings();

        OEDots dots = new OEDots(10000, 100, "molecules");

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            dots.Update();
            ringdict.AddRings(mol);
        }

        dots.Total();

        uint nrnewrings = ringdict.NumRings() - nrrings;
        OEChem.OEThrow.Info(nrnewrings.ToString() + " new ring templates have been added!
↪");

        OEChem.OEWrite2DRingDictionary(ordfname, ringdict);

        return 0;
    }
    private static string interfaceData = @"
!BRIEF [-in] <input> [-inringdict] <input ringdict> [-outringdict] <output ringdict>

!CATEGORY ""input/output options :""

  !PARAMETER -in
    !ALIAS -i
    !TYPE string
    !REQUIRED true
    !KEYLESS 1
    !VISIBILITY simple
    !BRIEF Input 2D molecule filename
```

(continues on next page)

(continued from previous page)

```
  !END

  !PARAMETER -inringdict
    !ALIAS -ird
    !TYPE string
    !REQUIRED true
    !KEYLESS 2
    !VISIBILITY simple
    !BRIEF Input ring dictionary OEB filename
    !DETAIL
        2D ring dictionaries can be generated by the following OEChem examples:
        C++    - createringdict.cpp
        Python - createringdict.py
        Java   - CreateRingDict.java
        C#     - CreateRingDict.cs
  !END

  !PARAMETER -outringdict
    !ALIAS -ord
    !TYPE string
    !REQUIRED true
    !KEYLESS 3
    !VISIBILITY simple
    !BRIEF Output ring dictionary OEB filename
  !END

!END
";
}
```

See also

- *OE2DRingDictionary* class
- *OEIsValid2DRingDictionary* function
- *OEWrite2DRingDictionary* function

Generate 2D coordinates with user-defined ring templates

A program that generates 2D coordinates with user-defined ring layouts.

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
```

(continues on next page)

(continued from previous page)

```
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
 * Generates 2D coordinates using user-defined ring templates
 *****************************************************************************/
using System;
using OpenEye.OEChem;

public class Generate2D
{
    public static int Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "Generate2D", args);

        string ifname = itf.GetString("-in");
        string ofname = itf.GetString("-out");

        oemolistream ifs = new oemolistream();
        if (!ifs.open(ifname))
        {
            OEChem.OEThrow.Fatal("Cannot open input file for reading!");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(ofname))
        {
            OEChem.OEThrow.Fatal("Cannot open output file for writing!");
        }

        if (!OEChem.OEIs2DFormat(ofs.GetFormat()))
        {
            OEChem.OEThrow.Fatal("Invalid output file format for 2D coordinates!");
        }

        if (itf.HasString("-ringdict"))
        {
            string rdfname = itf.GetString("-ringdict");

            if (!OEChem.OEIsValid2DRingDictionary(rdfname))
            {
                OEChem.OEThrow.Warning("Invalid 2D ring dictionary file!");
            }
            else
            {
```

(continues on next page)

(continued from previous page)

```
                OEChem.OEInit2DRingDictionary(rdfname);
            }
        }

        OEGraphMol mol = new OEGraphMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEGenerate2DCoordinates(mol);
            OEChem.OEWriteMolecule(ofs, mol);
        }

        return 0;
    }
    private static string interfaceData = @"
!BRIEF [-in] <input> [-out] <output> [-ringdict] <ringdict file>

!CATEGORY ""input/output options :""

  !PARAMETER -in
    !ALIAS -i
    !TYPE string
    !REQUIRED true
    !KEYLESS 1
    !VISIBILITY simple
    !BRIEF Input filename
  !END

  !PARAMETER -out
    !ALIAS -o
    !TYPE string
    !REQUIRED true
    !KEYLESS 2
    !VISIBILITY simple
    !BRIEF Output filename
  !END

  !PARAMETER -ringdict
    !ALIAS -rd
    !TYPE string
    !REQUIRED false
    !VISIBILITY simple
    !BRIEF Ring dictionary file
    !DETAIL
        2D ring dictionaries can be generated by the following OEChem examples:
        C++    - createringdict.cpp
        Python - createringdict.py
        Java   - CreateRingDict.java
        C#     - CreateRingDict.cs
  !END

!END
";
```

(continues on next page)

(continued from previous page)

```
}
```

See also

- *OE2DRingDictionary* class
- *OEWrite2DRingDictionary* function
- *OEInit2DRingDictionary* function

Print anisotropic B-factors

A program that prints anisotropic B-factors for each atom in a molecule that has those parameters. Useful for checking a PDB file for anisotropic B factors.

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
//****************************************************************************
// Output anisotropic B factor information
//****************************************************************************/
using System;
using OpenEye.OEChem;

public class Anisou
{
    public static void Main(string[] args)
    {
        OEInterface itf = new OEInterface(interfaceData, "Anisou", args);

        bool verbose  = itf.GetBool("-verbose");
        string ifname = itf.GetString("-input");

        oemolistream ims = new oemolistream();
        if (! ims.open(ifname))
```

(continues on next page)

(continued from previous page)

```
            OEChem.OEThrow.Fatal("Unable to open " + ifname + " for reading");

        ims.SetFlavor(OEFormat.PDB, OEIFlavor.PDB.Default|OEIFlavor.PDB.DATA);

        foreach(OEGraphMol mol in ims.GetOEGraphMols())
        {
            if (verbose)
            {
               if (! OEChem.OEHasResidues(mol))
                   OEChem.OEPerceiveResidues(mol, OEPreserveResInfo.All);

               foreach(OEAtomBase atom in mol.GetAtoms())
               {
                   OEResidue res = OEChem.OEAtomGetResidue(atom);

                   OEAnisoUij uij = new OEAnisoUij();
                   if (OEChem.OEGetAnisou(uij, atom))
                       OEChem.OEThrow.Info(mol.GetTitle() + " " +
                                           atom.GetName() +
                                           res.GetAlternateLocation() + " " +
                                           res.GetName() +
                                           res.GetResidueNumber() +
                                           res.GetInsertCode() + " " +
                                           res.GetChainID() + " (" +
                                           "u11=" + uij.GetU11() + ", " +
                                           "u22=" + uij.GetU22() + ", " +
                                           "u33=" + uij.GetU33() + ", " +
                                           "u12=" + uij.GetU12() + ", " +
                                           "u13=" + uij.GetU13() + ", " +
                                           "u23=" + uij.GetU23() + ")");
                   else
                       OEChem.OEThrow.Info(mol.GetTitle() + " " +
                                           atom.GetName() +
                                           res.GetAlternateLocation() + " " +
                                           res.GetName() +
                                           res.GetResidueNumber() +
                                           res.GetInsertCode() + " " +
                                           res.GetChainID() +
                                           " -no-anisou-");
               }
            }
            OEChem.OEThrow.Info(mol.GetTitle() + " " +
                                OEChem.OECount(mol, new OEHasAnisou()) +
                                " atoms with anisou data (out of " +
                                mol.NumAtoms() + ")");
        }
    }

    private static string interfaceData=@"
!BRIEF [-v] [-i] <mol file>

!CATEGORY ""input options""
```

(continues on next page)

(continued from previous page)

```
  !PARAMETER -input
    !ALIAS -i
    !TYPE string
    !REQUIRED true
    !BRIEF input mol file name
    !KEYLESS 1
  !END
!END

!CATEGORY ""options""

  !PARAMETER -verbose
    !ALIAS -v
    !TYPE bool
    !DEFAULT false
    !BRIEF verbose
  !END

!END
";
}
```

See also

- *OEHasAnisou* class
- *OEGetAnisou* function
- *OESetAnisou* function

Print dots

A program that enables dots to printed to the screen to show the progress of another example program.

Example

```
prompt> Convert input.mol2 .oeb.gz | Dots .oeb.gz output.oeb.gz
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
```

(continues on next page)

(continued from previous page)

```
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
* Show dots when piping example programs
*****************************************************************************/
using System;
using OpenEye.OEChem;

public class Dots
{
    public static void Main(string[] args)
    {
        if (args.Length < 2 || args.Length >3)
        {
            OEChem.OEThrow.Usage("Dots <infile> <outfile> [interval, default = 100]");
        }

        oemolistream ifs = new oemolistream();
        if (!ifs.open(args[0]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[0] + " for reading");
        }

        oemolostream ofs = new oemolostream();
        if (!ofs.open(args[1]))
        {
            OEChem.OEThrow.Fatal("Unable to open " + args[1] + " for writing");
        }

        OEDots dots = new OEDots(100, 10, "molecules");
        if (args.Length  == 3)
        {
            uint i = uint.Parse(args[2]);
            dots.SetBigStep(i);
            dots.SetSmallStep(i/10);
        }

        OEMol mol = new OEMol();
        while (OEChem.OEReadMolecule(ifs, mol))
        {
            OEChem.OEWriteMolecule(ofs, mol);
            dots.Update();
        }
    }
}
```

See also

- *OEDots* class

Define parameters

A program that shows definition of parameters.

Example

Note

This example uses Preliminary API that currently only available in C++ and Python.

Code

See also

- *OEOptions* class

Basic example of setting up an OE3DMolStyle on a molecule

A program that is an example for setting an *OE3DMolStyle* to a *OEMolBase*. The style being apply will label and color atoms by element. The atoms will be displayed in ball and stick format. The protein structures will be displayed as ribbons and colored based off residue. Hydrogen visibility is set to polar, and the interaction style is set to Inter molecular hydrogen bonds.

Example

Code

See also

- *OE3DMolStyle* class
- *OEMolStyleColorer* class

OE3DMolStyle focused on B-Factor styling

A program utilized to visualize a *OEMolBase* by B-Factor with a *OE3DMolStyle*. This example shows all ways to visualize B-Factor from atoms to protein ribbon coloring and atom labeling.

Example

Code

See also

- *OE3DMolStyle* class

- *OEMolStyleColorer* class

OE3DMolStyle hierarchy between molecule, conformers, and atoms

A program that shows how the style hierarchy in *OE3DMolStyle* is inherited between molecules, conformer, and atoms. The style applied to the main molecule will be the top level style. The style applied to each conformer will take priority over any overlapping style options between the two. The style applied to each atom will take priority over both the conformers and the molecule. The output file will have a molecule style of: Wireframe, blue, yellow molecular surface, and no hydrogens. The first conformer will have a style that indludes: CPK, blue, yellow molecular surface, and polar hydrogens. The second conformer will have the same style as the molecule. The first atom on the first conformer will have a style that includes: Stick, white, no surface, and polar hydrogens. The first atom on the second conformer will have a style that includes: Stick, white, no surface, and no hydrogens. The second atom on the first conformer will have a style that includes: CPK, pink, blue molecular surface, and polar H.

Example

Code

See also

- *OE3DMolStyle* class
- *OEMolStyleColorer* class

Creating an OECIFData Object

In this example, we will use the `OECIFData` to get, add, delete, and set header data for an input CIF file. We will use the PDB code 5NEV read using the mmCIF format, but any CIF header can be accessed with these APIs. First, the molecule is read and the OECIFData object is created. This object holds the header data for easy editing. To save any changes made to the header back onto the molecule, use the `OESetMMCIFChemCompData` function.

Read the data and generate the OECIFData object.

Full listing.

The OECIFData class allows access to all the data in a CIF header, but each CIF header category in a data block is broken into smaller `OECIFCategory` objects which can be accessed and modified.

Full listing.

For more information about OECIFData, OECIFCategory, and the CIF file format, see *CIF File Format*.

Attribute data is accessed by first specifying the CIF category name, and a specific data value in that category can be accessed with the attribute. For the following example, the category is '_audit_author.', and the values correlate with the attribute names.

Full listing.

Editing data items can be done using the Set, Add, and Delete OECIFData functions. When specifying a data value with multiple rows of data (i.e., CIF Loop Data), attribute values on a specific row can be accessed with a base-0 row index:

Full listing.

The `OECIFData` object can be used to update or synchronize header data with changes to the molecule. We can simulate this by first modifying a part of the molecule:

Full listing.

Updating requires the user to specify which parts of the header are to be updated. This is done using the *OECIFOptions* class. A more detailed explanation about which data items are affected by perception can be found in the *CIF File Format* theory. In the following example, we update the header and check to make sure that our residue changes from above have been applied to the pertinent header locations.

Full listing.

- *OESetMMCIFChemCompData* function
- *OEUpdateMMCIFData* function
- *OECIFData* class
- *OECIFCategory* class

Setting mmCIF Chemical Component Header

Use the reference molecule to populate or replace a residue's _chem_comp., _chem_comp_atom., and _chem_comp_bond. header information from the input CIF file.

Example

Code

See also

- *OESetMMCIFChemCompData* function
- *OECIFData* class

3.2 OEBio Examples

The following table lists the currently available **OEBio TK** examples:

Program	Description
Backbone	extracting protein backbone
ResHist	printing protein amino acids histogram
SplitMolComplex	splitting protein complex
SplitMolComplexLowLevel	splitting protein complex (low level)
SplitMolComplexFrags	splitting protein complex into fragments
ProteinPrep	protein prep
PrintInteractions	interaction perception

Examples:

3.2.1 Extracting Protein Backbone

A program that extracts the backbone from a protein. An example command would be:

```
prompt> Backbone input.pdb output.pdb
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
using System;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class Backbone
{
    public void MakeBackbone(string infile, string outfile)
    {
        oemolistream ifs = new oemolistream();
        ifs.SetFlavor(OEFormat.PDB, OEIFlavor.PDB.Default ^ OEIFlavor.PDB.DATA);
        if(!ifs.open(infile))
        {
            Console.WriteLine("Unable to open infile: " + infile);
            return;
        }
        oemolostream ofs = new oemolostream();
        if(!ofs.open(outfile))
        {
            Console.WriteLine("Unable to open outfile: " + outfile);
            return;
        }

        bool adjustHCount = true;
        foreach(OEGraphMol mol in ifs.GetOEGraphMols())
        {
            if (! OEChem.OEHasResidues(mol))
                OEChem.OEPerceiveResidues(mol, OEPreserveResInfo.All);
            OEAtomBaseIter atomiter = mol.GetAtoms(new OEIsBackboneAtom());
            OEIsAtomMember member = new OEIsAtomMember(atomiter);

            OEGraphMol backboneMol = new OEGraphMol();
            OEChem.OESubsetMol(backboneMol, mol, member, adjustHCount);
            OEChem.OEWriteMolecule(ofs, backboneMol);
```

(continues on next page)

(continued from previous page)

```
        }
        ofs.close();
    }

    public static void Main(string [] argv)
    {
        Backbone app = new Backbone();
        if(argv.Length != 2)
        {
            OEChem.OEThrow.Usage("Backbone <inmol> <outmol>");
        }
        app.MakeBackbone(argv[0], argv[1]);
    }
}
```

See also

- *OEPerceiveResidues* function
- *OEIsBackboneAtom* predicate
- *OESubsetMol* function

3.2.2 Extracting Ligand from *OEDesignUnit*

A program that extracts the ligand from an *OEDesignUnit*. See the C++ and python examples.

Code

3.2.3 Printing protein amino acids histograms

A program that prints a histogram of the occurrences of various amino acids in a protein. An example command would be:

```
prompt> ResHist input.pdb
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

(continues on next page)

(continued from previous page)

```
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
using System;
using System.Collections.Generic;
using System.Linq;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class ResHist
{
    public void ShowResHist(string infile)
    {
        oemolistream ifs = new oemolistream();
        if(!ifs.open(infile))
        {
            Console.WriteLine("Unable to open infile: " + infile);
            return;
        }

        uint molCt=0;
        Dictionary<string, uint> dict = new Dictionary<string, uint>();

        foreach(OEGraphMol mol in ifs.GetOEGraphMols())
        {
            ++molCt;
            dict.Clear();
            Console.WriteLine("=====================================");
            Console.WriteLine("Molecule: "+molCt+" ( "+mol.GetTitle()+" )");
            if (! OEChem.OEHasResidues(mol))
                OEChem.OEPerceiveResidues(mol, OEPreserveResInfo.All);

            OEHierView hv = new OEHierView(mol);
            int resCt=0;
            foreach (OEHierResidue res in hv.GetResidues())
            {
                ++resCt;
                string name = res.GetOEResidue().GetName();
                if(dict.ContainsKey(name))
                {
                    dict[name] = dict[name]+1;
                }
                else
                {
                    dict[name] = 1;
                }
            }

            List<string> list = dict.Keys.ToList();
            list.Sort();
```

(continues on next page)

(continued from previous page)

```
            foreach (string name in list)
            {
                uint count = dict[name];
                double percent = (100.0*count/resCt);
                Console.WriteLine("{0,3} {1,3} {2,4:N1} %\n",name, dict[name], percent);
            }

            Console.WriteLine();
        }
    }
    public static void Main(string [] argv)
    {
        ResHist app = new ResHist();
        if(argv.Length != 1)
        {
            OEChem.OEThrow.Usage("ResHist <inmol>");
        }
        app.ShowResHist(argv[0]);
    }
}
```

See also

- *OEPerceiveResidues* function
- *OEHierView* class
- *OEHierResidue* class
- *OEHierResidue.GetOEResidue* method

3.2.4 Splitting a Macro-molecular Complex

A program that splits components of a protein structure, returning separate *ligand, protein-complex, waters,* and *other* molecules.

An example command that limits the protein-complex output to just the binding site residues would be:

```
prompt> SplitMolComplex -separateresidues input.pdb output.oeb.gz
```

Help is available for all the supported *OESplitMolComplexOptions* parameters:

```
prompt> SplitMolComplex --help all
```

```
Complete parameter list
 Display options :
   -verbose : If true, return more verbose output

 SplitMolComplex options :
   -bindingsitenum : Select this binding site
   -covalentbondtreatment : Covalent bond treatment
   -covalentligand : Split covalent ligands
   -ligandfilter : Ligand filter category
```

(continues on next page)

(continued from previous page)

```
   -ligandname : Ligand name
   -maxsitedistance : Maximum distance to be associated with the binding site
   -maxsurfacedistance : Maximum distance to be associated with the protein
                         surface
   -modelnum : Select this NMR model number
   -proteinfilter : Protein filter category
   -separateresidues : Separate individual residues before selection
   -surfacewaters : Select surface waters
   -waterfilter : Water filter category

 input/output options :
   -in : Input molecule (usually a pdb file)
   -out : Output molecule (usually an oeb)
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
// Split a mol complex (a PDB structure, for example) into basic categories

using System;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class SplitMolComplex
{
    public static int Main(string[] argv)
    {
        OEInterface itf = new OEInterface(interfaceData);

        OEBio.OEConfigureSplitMolComplexOptions(itf);

        if (!OEChem.OEParseCommandLine(itf, argv, "SplitMolComplex"))
            OEChem.OEThrow.Fatal("Unable to interpret command line!");
```

(continues on next page)

(continued from previous page)

```
        string iname = itf.GetString("-in");
        string oname = itf.GetString("-out");

        oemolistream ims = new oemolistream();
        if (itf.GetUnsignedInt("-modelnum") != 1)
            ims.SetFlavor(OEFormat.PDB,
                          OEChem.OEGetDefaultIFlavor(OEFormat.PDB)
                                              & ~OEIFlavor.PDB.ENDM);
        if (!ims.open(iname))
            OEChem.OEThrow.Fatal("Cannot open input file!");

        oemolostream oms = new oemolostream();
        if (!oms.open(oname))
            OEChem.OEThrow.Fatal("Cannot open output file!");

        OEGraphMol inmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(ims, inmol))
            OEChem.OEThrow.Fatal("Unable to read molecule from " + iname);
        ims.close();

        OESplitMolComplexOptions opts = new OESplitMolComplexOptions();
        OEBio.OESetupSplitMolComplexOptions(opts, itf);

        if (itf.GetBool("-verbose"))
        {
            // don't bother counting sites unless we're going to print them
            uint numSites = OEBio.OECountMolComplexSites(inmol, opts);
            OEChem.OEThrow.SetLevel(OEErrorLevel.Verbose);
            OEChem.OEThrow.Verbose("sites " + numSites);
        }

        OEGraphMol lig   = new OEGraphMol();
        OEGraphMol prot  = new OEGraphMol();
        OEGraphMol wat   = new OEGraphMol();
        OEGraphMol other = new OEGraphMol();

        if (! OEBio.OESplitMolComplex(lig, prot, wat, other, inmol, opts))
            OEChem.OEThrow.Fatal("Unable to split mol complex from " + iname);

        if (! (lig.NumAtoms() == 0))
        {
          OEChem.OEThrow.Verbose("  lig " + lig.GetTitle());
          OEChem.OEWriteMolecule(oms, lig);
        }

        if (! (prot.NumAtoms() == 0))
        {
          OEChem.OEThrow.Verbose(" prot " + prot.GetTitle());
          OEChem.OEWriteMolecule(oms, prot);
        }

        if (! (wat.NumAtoms() == 0))
```

(continues on next page)

(continued from previous page)

```
        {
          OEChem.OEThrow.Verbose("  wat " + wat.GetTitle());
          OEChem.OEWriteMolecule(oms, wat);
        }

        if (! (other.NumAtoms() == 0))
        {
          OEChem.OEThrow.Verbose("other " + other.GetTitle());
          OEChem.OEWriteMolecule(oms, other);
        }

        oms.close();
        return 0;
    }

    private static string interfaceData = @"
!BRIEF <inmol> [<outmol>]

!CATEGORY ""input/output options :""
   !PARAMETER -in
      !ALIAS -i
      !TYPE string
      !BRIEF Input molecule (usually a pdb file)
      !VISIBILITY simple
      !REQUIRED true
      !KEYLESS 1
   !END

   !PARAMETER -out
      !ALIAS -o
      !TYPE string
      !DEFAULT splitmolcomplex.oeb.gz
      !BRIEF Output molecule (usually an oeb)
      !VISIBILITY simple
      !REQUIRED false
      !KEYLESS 2
   !END
!END

!CATEGORY ""Display options :""
   !PARAMETER -verbose
      !ALIAS -v
      !TYPE bool
      !DEFAULT false
      !BRIEF If true, return show molecule titles and number of binding sites
      !VISIBILITY simple
      !REQUIRED false
   !END
!END
";

}
```

See also

- *OESplitMolComplex* function
- *OESplitMolComplexOptions* object
- *OEConfigureSplitMolComplexOptions* function
- *OESplitMolComplexSetup* function

3.2.5 Splitting a Macro-molecular Complex Efficiently and Flexibly

A program that splits components of a protein structure, returning separate *ligand, protein-complex, waters,* and *other* molecules. This version uses the low-level API to fragment and then combine selected components.

An example command that limits the protein-complex output to just the binding site residues would be:

```
prompt> SplitMolComplexLowLevel -separateresidues input.pdb output.oeb.gz
```

Help is available for all the supported *OESplitMolComplexOptions* parameters:

```
prompt> SplitMolComplexLowLevel --help all
```

```
splitmolcomplexlowlevel <inmol> [<outmol>]
Complete parameter list
    Display options :
      -verbose : If true, show molecule titles and number of binding sites

    SplitMolComplex options :
      -bindingsitenum : Select this binding site
      -covalentbondtreatment : Covalent bond treatment
      -covalentcofactor : Split covalent cofactors
      -covalentligand : Split covalent ligands
      -ligandfilter : Ligand filter category
      -ligandname : Ligand name
      -maxsitedistance : Maximum distance to be associated with the binding site
      -maxsurfacedistance : Maximum distance to be associated with the protein
                            surface
      -modelnum : Select this NMR model number
      -proteinfilter : Protein filter category
      -separateresidues : Separate individual residues before selection
      -surfacewaters : Select surface waters
      -waterfilter : Water filter category

    input/output options :
      -in : Input molecule (usually a pdb file)
      -out : Output molecule (usually an oeb)
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
```

(continues on next page)

(continued from previous page)

```
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
// Split a mol complex (a PDB structure, for example) using the low-level api

using System;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class SplitMolComplexLowLevel
{
    public static int Main(string[] argv)
    {
        OEInterface itf = new OEInterface(interfaceData);

        OEBio.OEConfigureSplitMolComplexOptions(itf);

        if (!OEChem.OEParseCommandLine(itf, argv, "SplitMolComplexLowLevel"))
            OEChem.OEThrow.Fatal("Unable to interpret command line!");

        string iname = itf.GetString("-in");
        string oname = itf.GetString("-out");

        oemolistream ims = new oemolistream();
        if (itf.GetUnsignedInt("-modelnum") != 1)
            ims.SetFlavor(OEFormat.PDB,
                          OEChem.OEGetDefaultIFlavor(OEFormat.PDB)
                                        & ~OEIFlavor.PDB.ENDM);
        if (!ims.open(iname))
            OEChem.OEThrow.Fatal("Cannot open input file!");

        oemolostream oms = new oemolostream();
        if (!oms.open(oname))
            OEChem.OEThrow.Fatal("Cannot open output file!");

        OEGraphMol inmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(ims, inmol))
            OEChem.OEThrow.Fatal("Unable to read molecule from " + iname);
        ims.close();

        OESplitMolComplexOptions opts = new OESplitMolComplexOptions();
```

(continues on next page)

(continued from previous page)

```
        OEBio.OESetupSplitMolComplexOptions(opts, itf);

        OEAtomBondSetVector frags = new OEAtomBondSetVector();

        if (! OEBio.OEGetMolComplexFragments(frags, inmol, opts))
            OEChem.OEThrow.Fatal("Unable to split mol complex from " + iname);

        uint numSites = OEBio.OECountMolComplexSites(frags);

       if (itf.GetBool("-verbose"))
        {
            OEChem.OEThrow.SetLevel(OEErrorLevel.Verbose);
            OEChem.OEThrow.Verbose("sites " + numSites);
        }

        OEGraphMol lig   = new OEGraphMol();
        OEGraphMol prot  = new OEGraphMol();
        OEGraphMol wat   = new OEGraphMol();
        OEGraphMol other = new OEGraphMol();

        if (! OEBio.OECombineMolComplexFragments(lig, frags, opts, opts.
↪GetLigandFilter()))
            OEChem.OEThrow.Fatal("Unable to split ligand from " + iname);

         if (! OEBio.OECombineMolComplexFragments(prot, frags, opts, opts.
↪GetProteinFilter()))
            OEChem.OEThrow.Fatal("Unable to split protein complex from " + iname);

        if (! OEBio.OECombineMolComplexFragments(wat, frags, opts, opts.
↪GetWaterFilter()))
            OEChem.OEThrow.Fatal("Unable to split waters from " + iname);

        if (! OEBio.OECombineMolComplexFragments(other, frags, opts, opts.
↪GetOtherFilter()))
            OEChem.OEThrow.Fatal("Unable to split other mols from " + iname);

        if (! (lig.NumAtoms() == 0))
        {
          OEChem.OEThrow.Verbose("  lig " + lig.GetTitle());
          OEChem.OEWriteMolecule(oms, lig);
        }

        if (! (prot.NumAtoms() == 0))
        {
          OEChem.OEThrow.Verbose(" prot " + prot.GetTitle());
          OEChem.OEWriteMolecule(oms, prot);
        }

        if (! (wat.NumAtoms() == 0))
        {
          OEChem.OEThrow.Verbose("  wat " + wat.GetTitle());
          OEChem.OEWriteMolecule(oms, wat);
```

(continues on next page)

(continued from previous page)

```
        }

        if (! (other.NumAtoms() == 0))
        {
          OEChem.OEThrow.Verbose("other " + other.GetTitle());
          OEChem.OEWriteMolecule(oms, other);
        }

        oms.close();
        return 0;
    }

    private static string interfaceData = @"
!BRIEF <inmol> [<outmol>]

!CATEGORY ""input/output options :""
   !PARAMETER -in
      !ALIAS -i
      !TYPE string
      !BRIEF Input molecule (usually a pdb file)
      !VISIBILITY simple
      !REQUIRED true
      !KEYLESS 1
   !END

   !PARAMETER -out
      !ALIAS -o
      !TYPE string
      !DEFAULT splitmolcomplex.oeb.gz
      !BRIEF Output molecule (usually an oeb)
      !VISIBILITY simple
      !REQUIRED false
      !KEYLESS 2
   !END
!END

!CATEGORY ""Display options :""
   !PARAMETER -verbose
      !ALIAS -v
      !TYPE bool
      !DEFAULT false
      !BRIEF If true, return show molecule titles and number of binding sites
      !VISIBILITY simple
      !REQUIRED false
   !END
!END
";

}
```

See also

- *OEGetMolComplexFragments* function
- *OECombineMolComplexFragments* function
- *OESplitMolComplexOptions* object
- *OEConfigureSplitMolComplexOptions* function
- *OESplitMolComplexSetup* function

3.2.6 Split a Macro-molecular Complex Into Fragments

A program that extracts and filters components of a protein structure, returning individual fragments.

An example command that returns just the ligands for all binding sites would be:

```
prompt> SplitMolComplexFrags -waterfilter nothing -proteinfilter nothing -bindingsitenum␣
↪0 input.pdb output.sdf
```

Help is available for all the supported *OESplitMolComplexOptions* parameters:

```
prompt> SplitMolComplexFrags --help all
```

```
Complete parameter list
 Display options :
   -verbose : If true, return more verbose output

 SplitMolComplex options :
   -bindingsitenum : Select this binding site
   -covalentbondtreatment : Covalent bond treatment
   -covalentligand : Split covalent ligands
   -ligandfilter : Ligand filter category
   -ligandname : Ligand name
   -maxsitedistance : Maximum distance to be associated with the binding site
   -maxsurfacedistance : Maximum distance to be associated with the protein
                         surface
   -modelnum : Select this NMR model number
   -proteinfilter : Protein filter category
   -separateresidues : Separate individual residues before selection
   -surfacewaters : Select surface waters
   -waterfilter : Water filter category

 input/output options :
   -in : Input molecule (usually a pdb file)
   -out : Output molecule (usually an oeb)
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
```

(continues on next page)

(continued from previous page)

```
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
// Split a mol complex (a PDB structure, for example) into fragments

using System;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class SplitMolComplexFrags
{
    public static int Main(string[] argv)
    {
        OEInterface itf = new OEInterface(interfaceData);

        OEBio.OEConfigureSplitMolComplexOptions(itf);

        if (!OEChem.OEParseCommandLine(itf, argv, "SplitMolComplexFrags"))
            OEChem.OEThrow.Fatal("Unable to interpret command line!");

        string iname = itf.GetString("-in");
        string oname = itf.GetString("-out");

        oemolistream ims = new oemolistream();
        if (itf.GetUnsignedInt("-modelnum") != 1)
            ims.SetFlavor(OEFormat.PDB,
                          OEChem.OEGetDefaultIFlavor(OEFormat.PDB)
                                          & ~OEIFlavor.PDB.ENDM);
        if (!ims.open(iname))
            OEChem.OEThrow.Fatal("Cannot open input file!");

        oemolostream oms = new oemolostream();
        if (!oms.open(oname))
            OEChem.OEThrow.Fatal("Cannot open output file!");

        OEGraphMol inmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(ims, inmol))
            OEChem.OEThrow.Fatal("Unable to read molecule from " + iname);
        ims.close();

        OESplitMolComplexOptions opts = new OESplitMolComplexOptions();
```

(continues on next page)

(continued from previous page)

```
        OEBio.OESetupSplitMolComplexOptions(opts, itf);

        if (itf.GetBool("-verbose"))
        {
            // don't bother counting sites unless we're going to print them
            uint numSites = OEBio.OECountMolComplexSites(inmol, opts);
            OEChem.OEThrow.SetLevel(OEErrorLevel.Verbose);
            OEChem.OEThrow.Verbose("sites " + numSites);
        }

        foreach (OEMolBase frag in OEBio.OEGetMolComplexComponents(inmol, opts))
        {
          OEChem.OEThrow.Verbose("frag " + frag.GetTitle());
          OEChem.OEWriteMolecule(oms, frag);
        }

        oms.close();
        return 0;
    }

    private static string interfaceData = @"
!BRIEF <inmol> [<outmol>]

!CATEGORY ""input/output options :""
   !PARAMETER -in
      !ALIAS -i
      !TYPE string
      !BRIEF Input molecule (usually a pdb file)
      !VISIBILITY simple
      !REQUIRED true
      !KEYLESS 1
   !END

   !PARAMETER -out
      !ALIAS -o
      !TYPE string
      !DEFAULT splitmolcomplex.oeb.gz
      !BRIEF Output molecule (usually an oeb)
      !VISIBILITY simple
      !REQUIRED false
      !KEYLESS 2
   !END
!END

!CATEGORY ""Display options :""
   !PARAMETER -verbose
      !ALIAS -v
      !TYPE bool
      !DEFAULT false
      !BRIEF If true, return show fragment titles and number of binding sites
      !VISIBILITY simple
      !REQUIRED false
```

(continues on next page)

(continued from previous page)

```
   !END
!END
";
}
```

See also

- *OEGetMolComplexComponents* function
- *OESplitMolComplexOptions* object
- *OEConfigureSplitMolComplexOptions* function
- *OESplitMolComplexSetup* function

3.2.7 Preparing a Protein

A program that selects the highest occupancy alternate locations of a protein, adds hydrogens in orientations that make appropriate hydrogen bonds, and splits out the complex for the selected binding site (optionally separating the ligand from the protein complex). This is an initial prototype of a "protein prep" workflow.

An example command that splits out the ligand and limits water sampling is as follows:

```
prompt> ProteinPrep -waterprocessing ignore input.pdb prot.oeb.gz lig.oeb.gz
```

Help is available for all the supported parameters:

```
prompt> ProteinPrep --help all
```

```
proteinprep [-options] <inmol> [<outcplx> [<outlig>]]
Complete parameter list
    SplitMolComplex options :
      -bindingsitenum : Select this binding site
      -covalentligand : Split covalent ligands
      -ligandfilter : Ligand filter category
      -ligandname : Ligand name
      -maxsitedistance : Maximum distance to be associated with the binding site
      -maxsurfacedistance : Maximum distance to be associated with the protein
                            surface
      -modelnum : Select this NMR model number
      -proteinfilter : Protein filter category
      -separateresidues : Separate individual residues before selection
      -surfacewaters : Select surface waters
      -waterfilter : Water filter category

    input/output options :
      -in : Input molecule filename (must have 3D coordinates)
      -cplxout : Output complex filename
      -ligout : Output ligand filename

    Calculation options :
      -alts : Alternate location atom handling (affects atom:atom interactions)
      -placehydrogens : If false, hydrogens will not be added
```

(continues on next page)

(continued from previous page)

```
    -waterprocessing : How waters are processed (can greatly affect runtime)
    -standardizehyd : If false, bonds for hydrogens are not adjusted to standard
                      lengths
    -clashcutoff : Van der Waals overlap (in Angstroms) defined to be a bad
                   clash
    -flipbias : Scale factor for the bias against flipping sidechains such as
                HIS

  Display options :
    -verbose : Display more information about the process
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
// Prepare a molecule: alts, hydrogens, split ligand

using System;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class ProteinPrep
{
    static uint WaterProcess(string processName)
    {
        if (processName.Equals("fullsearch"))
            return OEPlaceHydrogensWaterProcessing.FullSearch;
        else if (processName.Equals("focused"))
            return OEPlaceHydrogensWaterProcessing.Focused;
        return OEPlaceHydrogensWaterProcessing.Ignore;
    }

    public static int Main(string[] argv)
    {
        OEInterface itf = new OEInterface(interfaceData);
```

(continues on next page)

(continued from previous page)

```
        OEBio.OEConfigureSplitMolComplexOptions(itf,
                                    OESplitMolComplexSetup.All              &
                                ~ ( OESplitMolComplexSetup.CovBondTreatment |
                                    OESplitMolComplexSetup.CovCofactor ) );

        if (! OEChem.OEParseCommandLine(itf, argv, "ProteinPrep"))
            OEChem.OEThrow.Fatal("Unable to interpret command line!");

        bool verbose = itf.GetBool("-verbose");
        if (verbose)
            OEChem.OEThrow.SetLevel(OEErrorLevel.Verbose);

        string altProcess = itf.GetString("-alts");
        bool keepAlts     = ! altProcess.Equals("a");
        bool highestOcc   =  (altProcess.Equals("occupancy"));
        bool compareAlts  =  (altProcess.Equals("compare"));

        uint siteNum    = itf.GetUnsignedInt("-bindingsitenum");
        bool allSites   = (siteNum == 0);
        bool otherModel = (itf.GetUnsignedInt("-modelnum") != 1);

        bool placeHyd = itf.GetBool("-placehydrogens");
        bool splitlig = itf.HasString("-ligout");

        string watProcessName = itf.GetString("-waterprocessing");
        uint waterProcess = WaterProcess(watProcessName);

        bool standardize = itf.GetBool("-standardizehyd");
        double badclash  = itf.GetDouble("-clashcutoff");
        double flipbias  = itf.GetDouble("-flipbias");
        double maxStates = itf.GetDouble("-maxsubstates");

        uint flavor = OEIFlavor.PDB.Default | OEIFlavor.PDB.DATA;
        if (keepAlts)
          flavor |= OEIFlavor.PDB.ALTLOC;
        if (otherModel)
          flavor &= ~ OEIFlavor.PDB.ENDM;

        oemolistream ims = new oemolistream();
        ims.SetFlavor(OEFormat.PDB, flavor);

        string inputFile = itf.GetString("-in");
        if (! ims.open(inputFile))
            OEChem.OEThrow.Fatal("Unable to open " +
                                 inputFile + " for reading.");

        if (! OEChem.OEIs3DFormat(ims.GetFormat()))
            OEChem.OEThrow.Fatal(inputFile + " is not in a 3D format.");

        uint inftype=OEChem.OEGetFileType(OEChem.OEGetFileExtension(inputFile));
        if ((inftype == OEFormat.PDB) && (! keepAlts))
```

(continues on next page)

(continued from previous page)

```
            OEChem.OEThrow.Verbose("Default processing of alt locations " +
                                   "(keep just 'A' and ' ').");

        OESplitMolComplexOptions sopt = new OESplitMolComplexOptions();
        OEBio.OESetupSplitMolComplexOptions(sopt, itf);

        OEGraphMol inmol = new OEGraphMol();
        if (! OEChem.OEReadMolecule(ims, inmol))
            OEChem.OEThrow.Fatal("Unable to read " + inputFile + ".");

        ims.close();

        if (inmol.NumAtoms() == 0)
            OEChem.OEThrow.Fatal("Input molecule " + inputFile + " contains no atoms.");

        if (inmol.GetTitle().Equals(""))
            inmol.SetTitle("input mol");

        OEChem.OEThrow.Verbose("Processing " + inmol.GetTitle() + ".");

        if (! OEChem.OEHasResidues(inmol))
            OEChem.OEPerceiveResidues(inmol, OEPreserveResInfo.All);

        if (highestOcc || compareAlts)
        {
            OEAltLocationFactory alf = new OEAltLocationFactory(inmol);
            if (alf.GetGroupCount() != 0)
            {
                if (highestOcc)
                {
                    OEChem.OEThrow.Verbose("Dropping alternate locations " +
                                           "from input.");
                    alf.MakePrimaryAltMol(inmol);
                }
                else if (compareAlts)
                {
                    OEChem.OEThrow.Verbose("Fixing alternate location issues.");
                    inmol = new OEGraphMol(alf.GetSourceMol());
                }
            }
        }

        OEGraphMol outmol = new OEGraphMol();
        if (allSites)
            outmol = inmol;
        else
        {
          OEChem.OEThrow.Verbose("Splitting out selected complex.");

          OESplitMolComplexOptions soptSiteSel =
                                            new OESplitMolComplexOptions(sopt);
          soptSiteSel.SetSplitCovalent(false); // do any cov lig splitting later
```

(continues on next page)

(continued from previous page)

```
        OEAtomBondSetVector frags = new OEAtomBondSetVector();
        if (! OEBio.OEGetMolComplexFragments(frags, inmol, soptSiteSel))
            OEChem.OEThrow.Fatal("Unable to fragment " +
                                 inmol.GetTitle() + ".");

        uint howManySites = OEBio.OECountMolComplexSites(frags);
        if (howManySites < siteNum)
        {
          OEChem.OEThrow.Warning("Binding site count (" + howManySites +
                                 ") less than requested site (" + siteNum +
                                 ") in " + inmol.GetTitle() + ".");
          return 0;
        }

        if (! OEBio.OECombineMolComplexFragments(outmol, frags, soptSiteSel))
            OEChem.OEThrow.Fatal("Unable to collect fragments from " +
                                 inmol.GetTitle() + ".");

        if (outmol.NumAtoms() == 0)
            OEChem.OEThrow.Fatal("No fragments selected from " +
                                 inmol.GetTitle() + ".");
      }

      if (placeHyd)
      {
          OEChem.OEThrow.Verbose("Adding hydrogens to complex.");

          OEPlaceHydrogensOptions hopt = new OEPlaceHydrogensOptions();
          hopt.SetAltsMustBeCompatible(compareAlts);
          hopt.SetStandardizeBondLen(standardize);
          hopt.SetWaterProcessing(waterProcess);
          hopt.SetBadClashOverlapDistance(badclash);
          hopt.SetFlipBiasScale(flipbias);
          hopt.SetMaxSsubstateCutoff(maxStates);

          if (verbose)
          {
              OEPlaceHydrogensDetails details = new OEPlaceHydrogensDetails();
              if (! OEBio.OEPlaceHydrogens(outmol, details, hopt))
                  OEChem.OEThrow.Fatal("Unable to place hydrogens and get details on "␣
↪+
                                       inmol.GetTitle() + ".");
              OEChem.OEThrow.Verbose(details.Describe());
          }
          else
          {
              if (! OEBio.OEPlaceHydrogens(outmol, hopt))
                  OEChem.OEThrow.Fatal("Unable to place hydrogens on " +
                                       inmol.GetTitle() + ".");
          }
      }
```

(continues on next page)

(continued from previous page)

```
        oemolostream oms1 = new oemolostream();
        string cplxFile = itf.GetString("-cplxout");
        if (! oms1.open(cplxFile))
            OEChem.OEThrow.Fatal("Unable to open " +
                                 cplxFile + " for writing.");

        if (splitlig)
        {
            OEChem.OEThrow.Verbose("Splitting ligand from complex.");

            uint returnAllSites = 0;
            OEBio.OESetupSplitMolComplexOptions(sopt, itf, returnAllSites);

            OEAtomBondSetVector frags = new OEAtomBondSetVector();
            if (! OEBio.OEGetMolComplexFragments(frags, outmol, sopt))
                OEChem.OEThrow.Fatal("Unable to fragment complex from " +
                                     inmol.GetTitle() + ".");

            OEUnaryRoleSetPred lfilter = sopt.GetLigandFilter();

            OEGraphMol protComplex = new OEGraphMol();

            if (! OEBio.OECombineMolComplexFragments(protComplex,
                                                     frags,
                                                     sopt,
                                                     new OENotRoleSet(lfilter)))
                OEChem.OEThrow.Fatal("Unable to collect complex from " +
                                     inmol.GetTitle() + ".");

            if (protComplex.NumAtoms() == 0)
                OEChem.OEThrow.Warning("No complex identified in " +
                                       inmol.GetTitle() + ".");
            else
                OEChem.OEWriteMolecule(oms1, protComplex);

            OEGraphMol lig = new OEGraphMol();

            if (! OEBio.OECombineMolComplexFragments(lig, frags, sopt, lfilter))
                OEChem.OEThrow.Fatal("Unable to collect ligand from " +
                                     inmol.GetTitle() + ".");

            if (lig.NumAtoms() == 0)
                OEChem.OEThrow.Warning("No ligand identified in " +
                                       inmol.GetTitle() + ".");
            else
            {
                oemolostream oms2 = new oemolostream();
                if (splitlig)
                {
                    string ligFile = itf.GetString("-ligout");
                    if (! oms2.open(ligFile))
```

(continues on next page)

(continued from previous page)

```
                        OEChem.OEThrow.Fatal("Unable to open " +
                                             ligFile + " for writing.");
                }
                OEChem.OEThrow.Verbose("Ligand: " + lig.GetTitle());
                OEChem.OEWriteMolecule(oms2, lig);
                oms2.close();
            }
        }
        else
            OEChem.OEWriteMolecule(oms1, outmol);

        oms1.close();

        return 0;
    }

    private static string interfaceData = @"
!BRIEF <inmol> [<outcplx> [<outlig>]]

!CATEGORY ""input/output options :"" 1
   !PARAMETER -in 1
      !ALIAS -i
      !TYPE string
      !BRIEF Input molecule filename (must have 3D coordinates)
      !SIMPLE true
      !REQUIRED true
      !KEYLESS 1
   !END

   !PARAMETER -cplxout 2
      !ALIAS -p
      !TYPE string
      !DEFAULT proteinprep.oeb.gz
      !BRIEF Output complex filename
      !SIMPLE true
      !REQUIRED false
      !KEYLESS 2
   !END

   !PARAMETER -ligout 3
      !ALIAS -l
      !TYPE string
      !BRIEF Output ligand filename
      !SIMPLE true
      !REQUIRED false
      !KEYLESS 3
   !END
!END

!CATEGORY ""Calculation options :"" 2
    !PARAMETER -alts 1
       !TYPE string
```

(continues on next page)

(continued from previous page)

```
      !LEGAL_VALUE occupancy
      !LEGAL_VALUE a
      !LEGAL_VALUE ignore
      !LEGAL_VALUE compare
      !DEFAULT occupancy
      !BRIEF Alternate location atom handling (affects atom:atom interactions)
      !SIMPLE true
      !REQUIRED false
      !DETAIL
        occupancy - keep just the highest average occupancy for each alt group
        a - keep only loc code A (and blank)
        ignore - assume alts already selected appropriately
        compare - keep all alts but only interact if same loc code (or blank)
   !END

   !PARAMETER -placehydrogens 2
      !TYPE bool
      !DEFAULT true
      !BRIEF If false, hydrogens will not be added
      !SIMPLE true
      !REQUIRED false
   !END

   !PARAMETER -waterprocessing 3
      !TYPE string
      !LEGAL_VALUE ignore
      !LEGAL_VALUE focused
      !LEGAL_VALUE fullsearch
      !DEFAULT fullsearch
      !BRIEF How waters are processed
      !SIMPLE true
      !REQUIRED false
      !DETAIL
        ignore - leave water hydrogens in a random orientation
        focused - search orientations based on neighboring polar groups
        fullsearch - do an extensive search of water orientations
   !END

   !PARAMETER -standardizehyd 4
      !ALIAS -stdhyd
      !TYPE bool
      !DEFAULT true
      !BRIEF If false, bonds for hydrogens are not adjusted to standard lengths
      !SIMPLE false
      !REQUIRED false
   !END

   !PARAMETER -clashcutoff 5
       !TYPE double
       !DEFAULT 0.4
       !BRIEF Van der Waals overlap (in Angstroms) defined to be a bad clash
       !SIMPLE false
```

(continues on next page)

(continued from previous page)

```
        !REQUIRED false
    !END

    !PARAMETER -flipbias 6
        !TYPE double
        !DEFAULT 1.0
        !BRIEF Scale factor for the bias against flipping sidechains such as HIS
        !SIMPLE false
        !REQUIRED false
    !END

    !PARAMETER -maxsubstates 7
        !TYPE double
        !DEFAULT 1.0e8
        !BRIEF Maximum number of substates in a single step of hydrogen placement␣
↪optimization
        !SIMPLE false
        !REQUIRED false
    !END
!END

!CATEGORY ""Display options :"" 9
   !PARAMETER -verbose 1
      !ALIAS -v
      !TYPE bool
      !DEFAULT false
      !BRIEF Display more information about the process
      !SIMPLE true
      !REQUIRED false
   !END
!END
";

}
```

See also

- *Protein Preparation* section
- *OEAltLocationFactory* object
- *OEIFlavor.PDB.ALTLOC* flavor
- *OEGetMolComplexFragments* function
- *OECombineMolComplexFragments* function
- *OESplitMolComplexOptions* object
- *OEConfigureSplitMolComplexOptions* function
- *OESplitMolComplexSetup* function
- *OEPlaceHydrogens* function
- *OEPlaceHydrogensOptions* object

- *OEPlaceHydrogensDetails* object

3.2.8 Perceive and Print Protein-Ligand Interactions

The program that perceives protein-ligand interactions and prints them out.

An example command would be:

```
prompt> PrintInteractions complex.pdb
```

Code

```
/*
(C) 2022 Cadence Design Systems, Inc. (Cadence)
All rights reserved.
TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is
provided to current licensees or subscribers of Cadence products or
SaaS offerings (each a "Customer").
Customer is hereby permitted to use, copy, and modify the Sample Code,
subject to these terms. Cadence claims no rights to Customer's
modifications. Modification of Sample Code is at Customer's sole and
exclusive risk. Sample Code may require Customer to have a then
current license or subscription to the applicable Cadence offering.
THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED.  OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be
liable for any damages or liability in connection with the Sample Code
or its use.
*/
/*****************************************************************************
 * Print protein-ligand interactions
 *****************************************************************************/

using System;
using System.Collections.Generic;
using OpenEye.OEChem;
using OpenEye.OEBio;

public class PrintInteractions
{
    private static OEInteractionHintComponentTypeBase ligtype = new␣
↪OELigandInteractionHintComponent();
    private static OEInteractionHintComponentTypeBase protype = new␣
↪OEProteinInteractionHintComponent();

    public static void Main(string[] argv)
    {
        OEInterface itf = new OEInterface(interfaceData);
        OEBio.OEConfigureSplitMolComplexOptions(itf, OESplitMolComplexSetup.LigName);

        if (!OEChem.OEParseCommandLine(itf, argv, "PrintInteractions"))
```

(continues on next page)

(continued from previous page)

```
            OEChem.OEThrow.Fatal("Unable to interpret command line!");

        string iname = itf.GetString("-complex");

        oemolistream ifs = new oemolistream();
        if (!ifs.open(iname))
            OEChem.OEThrow.Fatal("Cannot open input file!");

        OEGraphMol complexmol = new OEGraphMol();
        if (!OEChem.OEReadMolecule(ifs, complexmol))
            OEChem.OEThrow.Fatal("Unable to read molecule from " + iname);
        ifs.close();

        if (!OEChem.OEHasResidues(complexmol))
            OEChem.OEPerceiveResidues(complexmol, OEPreserveResInfo.All);

        // Separate ligand and protein

        OESplitMolComplexOptions sopts = new OESplitMolComplexOptions();
        OEBio.OESetupSplitMolComplexOptions(sopts, itf);

        OEGraphMol lig = new OEGraphMol();
        OEGraphMol prot = new OEGraphMol();
        OEGraphMol water = new OEGraphMol();
        OEGraphMol other = new OEGraphMol();

        OEUnaryRoleSetPred pfilter = sopts.GetProteinFilter();
        OEUnaryRoleSetPred wfilter = sopts.GetWaterFilter();
        sopts.SetProteinFilter(new OEOrRoleSet(pfilter, wfilter));
        sopts.SetWaterFilter(OEBio.OEMolComplexFilterFactory(OEMolComplexFilterCategory.
↪Nothing));

        OEBio.OESplitMolComplex(lig, prot, water, other, complexmol, sopts);

        if (lig.NumAtoms() == 0)
            OEChem.OEThrow.Fatal("Cannot separate complex!");

        // Perceive interactions

        OEInteractionHintContainer asite = new OEInteractionHintContainer(prot, lig);
        if (!OEBio.OEIsValidActiveSite(asite))
            OEChem.OEThrow.Fatal("Cannot initialize active site!");

        OEBio.OEPerceiveInteractionHints(asite);

        Console.WriteLine("Number of interactions: " + asite.NumInteractions());
        foreach (OEInteractionHintTypeBase itype in OEBio.
↪OEGetActiveSiteInteractionHintTypes())
        {
            Console.WriteLine(itype.GetName() + " :");

            foreach (OEInteractionHint inter in asite.GetInteractions(new␣
```

(continues on next page)

(continued from previous page)

```
↪OEHasInteractionHintType(itype)))
            {
                printInteraction(inter);
            }
        }
        Console.WriteLine("\nResidue interactions:");
        OEMolBase protein = asite.GetMolecule(new OEProteinInteractionHintComponent());
        foreach (OEResidue residue in OEBio.OEGetResidues(protein))
        {
            printResidueInteraction(asite, residue);
        }
        Console.WriteLine("\nLigand atom interactions:");
        OEMolBase ligand = asite.GetMolecule(new OELigandInteractionHintComponent());
        foreach (OEAtomBase atom in ligand.GetAtoms())
        {
            printLigandAtomInteractions(asite, atom);
        }
    }

    public static String getResidueName(OEResidue residue)
    {
        String name = String.Format("{0,3} {1,4:D} {2}", residue.GetName(),
                                                         residue.GetResidueNumber(),
                                                         residue.GetChainID());
        return name;
    }

    public static String getAtomName(OEAtomBase atom)
    {
        String name = String.Format("{0,2:D} {1}",
            atom.GetIdx(),
            OEChem.OEGetAtomicSymbol(atom.GetAtomicNum()));
        return name;
    }

    public static void printFragment(OEInteractionHintFragment frag) {
        OEInteractionHintComponentTypeBase fragtype = frag.GetComponentType();
        if (fragtype.Equals(ligtype))
        {
            Console.Write("ligand: ");
        }
        else if (fragtype.Equals(protype))
        {
            Console.Write("protein: ");
        }
        foreach (OEAtomBase atom in frag.GetAtoms())
        {
            Console.Write(getAtomName(atom) + " ");
        }
    }

    public static void printInteraction(OEInteractionHint inter)
```

(continues on next page)

(continued from previous page)

```
    {
        OEInteractionHintFragment bgnfrag = inter.GetBgnFragment();
        if (bgnfrag != null)
        {
            printFragment(bgnfrag);
        }
        OEInteractionHintFragment endfrag = inter.GetEndFragment();
        if (endfrag != null)
        {
            printFragment(endfrag);
        }
        Console.WriteLine("");
    }

    public static void printResidueInteraction(OEInteractionHintContainer asite,␣
↪OEResidue residue)
    {
        HashSet<OEAtomBase> ligatoms = new HashSet<OEAtomBase>();
        OEHasResidueInteractionHint ipred = new OEHasResidueInteractionHint(residue);
        foreach (OEInteractionHint inter in asite.GetInteractions(ipred))
        {
            OEInteractionHintFragment ligfrag = inter.GetFragment(new␣
↪OELigandInteractionHintComponent());
            if ((object)ligfrag != null)
            {
                foreach (OEAtomBase atom in ligfrag.GetAtoms())
                {
                    ligatoms.Add(atom);
                }
            }
        }
        if (ligatoms.Count > 0)
        {
            Console.Write(getResidueName(residue) + " :  ");
            foreach (OEAtomBase atom in ligatoms)
            {
                Console.Write(getAtomName(atom) + " ");
            }
            Console.WriteLine("");
        }
    }

    public static void printLigandAtomInteractions(OEInteractionHintContainer asite,␣
↪OEAtomBase atom)
    {
        HashSet<String> resnames = new HashSet<String>();
        OEHasInteractionHint ipred = new OEHasInteractionHint(atom);
        foreach (OEInteractionHint inter in asite.GetInteractions(ipred))
        {
            OEInteractionHintFragment profrag = inter.GetFragment(new␣
↪OEProteinInteractionHintComponent());
            if ((object)profrag != null)
```

(continues on next page)

(continued from previous page)

```
            {
                foreach (OEAtomBase patom in profrag.GetAtoms())
                {
                    OEResidue res = OEChem.OEAtomGetResidue(patom);
                    resnames.Add(getResidueName(res));
                }
            }
        }
        if (resnames.Count > 0)
        {
            Console.Write(getAtomName(atom) + " : ");
            foreach (String resname in resnames)
            {
                Console.Write(resname + " ");
            }
            Console.WriteLine("");
        }
    }
    private static string interfaceData = @"
!BRIEF [-complex] <input>

!CATEGORY input/output options :

  !PARAMETER -complex
    !ALIAS -c
    !TYPE string
    !KEYLESS 1
    !REQUIRED true
    !VISIBILITY simple
    !BRIEF Input filename of the protein complex
  !END

!END
";

}
```

See also

- *OESplitMolComplex* function
- *OEInteractionHintContainer* class
- *OEGetActiveSiteInteractionHintTypes* function
- *OEInteractionHint* and *OEInteractionHintFragment* classes
- *OEIsValidActiveSite* function
- *OEPerceiveInteractionHints* function
- *OEGetResidues* function
- *OELigandInteractionHintComponent* and *OEProteinInteractionHintComponent* classes

- *OEHasResidueInteractionHint* and *OEHasInteractionHint* predicate classes

3.3 OEMath Examples

The following table lists the currently available **OEMath** examples:

3.3.1 Fitting model with kernel PLS

The following code example shows how to fit a kernel PLS model. The input descriptors matrix must be symmetric for fitting with kernel PLS.

See also

- *OEKernelPLS* class

3.4 Python Cookbook Examples

There are large number of code examples available in OpenEye Python Cookbook that use **OpenEye Toolkits** to solve a wide range of cheminformatics and molecular modeling problems.

- 2D Depiction chapter contains examples that illustrate how the **OEDepict TK** and **Grapheme TK** can be utilized to depict molecules and visualize properties calculated with other OpenEye toolkits.
- Visualizing 3D Information chapter contains examples that illustrate how the **Grapheme TK** can be utilized to project complex 3D information into the 2D molecular graph.
- Cheminformatics chapter contains examples that solve various cheminformatics problems such as similarity search, ring perception and manipulating molecular graphs.
- Peptide Informatics chapter contains examples that solve various peptide informatics problems such as generating custom monomer sets, parsing and generating HELM representations, and visualizing peptides.

OpenEye Python Cookbook examples using OEChem TK

- Manipulating Large Molecule Files
- Enumerating Fragment Combinations
- Ring Perception
- Enumerating Atom Substitutions
- Reordering CSV File

OpenEye Python Cookbook examples related to peptide informatics using OEChem TK

- Generating Custom Monomer Set
- Convert HELM to SMILES
- Convert HELM File to Molecule File
- Convert SMILES to HELM

- Convert Molecule File to HELM File
- Generate Random Peptides
- Mutate Peptide

OpenEye Python Cookbook examples related to protein-ligand interactions using OEBio TK

- Accessing Interaction Hint Information
- Print Protein-Ligand Interactions to Console
- Print the Summary of Protein-Ligand Interactions for a Set of Molecules

CHAPTER

FOUR

API

4.1 OEChem API

4.1.1 OEChem Classes

OE2DRingDictionary

```
class OE2DRingDictionary
```

This class represents *OE2DRingDictionary* that stores ring system layouts *i.e.* templates that can be plugged into the 2D coordinate generation system.

See also

- *2D Coordinate Generation* chapter
- *OE2DRingDictionaryCreatorOptions* class
- `OEGenerate2DCoordinates` function
- `OEWrite2DRingDictionary` function

The following **OEChem TK** examples illustrate how to create and utilize user-defined ring layouts:

- Example program *CreateRingDict.cs*
- Example program *AppendRingDict.cs*
- Example program *Generate2D.cs*

The following **OEDepict TK** example shows how to generate a multi-page PDF report of user-defined ring layouts:

Constructors

```
OE2DRingDictionary()
```

Default constructor that creates an empty ring template dictionary with default parameters.

```
OE2DRingDictionary(const OE2DRingDictionaryCreatorOptions& opts)
```

Constructs an empty ring dictionary with the given parameters.

See also

- *OE2DRingDictionaryCreatorOptions* class

```
OE2DRingDictionary(oemolistream& ims)
OE2DRingDictionary(const std::string& filename)
```

Initializes a ring dictionary from a binary file written by the *OEWrite2DRingDictionary* function.

AddRings

```
unsigned int AddRings(const OEMolBase& mol)
```

Returns the number of ring systems extracted from the given molecule and adds them to the dictionary.

mol
The molecule **with 2D** coordinates.

By default, the rings are extracted and added to the dictionary by the following rules:

- A ring system will be added to the dictionary even if a corresponding layout already exists in the built-in ring dictionary *i.e.* user-defined layout will have higher priority.
- If a ring system already exists in the given dictionary then the new ring template will be ignored *i.e* first-come first-register.
- If a ring system has specified cis/trans bond stereo configuration then two layouts could be added to the ring dictionary: one with the cis/trans bond stereo configuration and one pure carbon skeleton framework without any bond stereo configuration.

The rules above can be altered when the ring dictionary is constructed via the *OE2DRingDictionaryCreatorOptions* class.

NumRings

```
unsigned int NumRings() const
```

Returns the number of rings stored in the dictionary.

OE2DRingDictionaryCreatorOptions

```
class OE2DRingDictionaryCreatorOptions
```

This class represents the *OE2DRingDictionaryCreatorOptions* class that encapsulates properties that determine how a ring template is added to a ring template dictionary (*OE2DRingDictionary*).

See also

- *2D Coordinate Generation* chapter
- *OE2DRingDictionary* class
- *OEWrite2DRingDictionary* function

Constructors

```
OE2DRingDictionaryCreatorOptions()
```

Default constructor.

An *OE2DRingDictionaryCreatorOptions* object is initialized with the following default parameters:

- A ring system will be added to the dictionary even if a corresponding layout already exists in the built-in ring dictionary *i.e.* user-defined layout will have higher priority.

  See also *OE2DRingDictionaryCreatorOptions.SetRetainExistingBuiltInTemplates* method.

- If a ring system already exists in the given dictionary then the new ring template will be ignored *i.e* first-come first-register.

  See also *OE2DRingDictionaryCreatorOptions.SetRetainExistingTemplates* method.

- If a ring system has specified cis/trans bond stereo configuration then two layouts could be added to the ring dictionary: one with the cis/trans bond stereo configuration and one pure carbon skeleton without any bond stereo configuration.

  See also *OE2DRingDictionaryCreatorOptions.SetAddCarbonSkeleton* and *OE2DRingDictionaryCreatorOptions.SetAddCarbonSkeletonWithBondStereo* methods.

```
OE2DRingDictionaryCreatorOptions(const OE2DRingDictionaryCreatorOptions& rhs)
```

Copy constructor.

GetAddCarbonSkeleton

```
bool GetAddCarbonSkeleton() const
```

Returns whether the carbon skeleton of a ring system is added to the dictionary. The default value is `true`.

See also

- *OE2DRingDictionaryCreatorOptions.SetAddCarbonSkeleton* method

GetAddCarbonSkeletonWithBondStereo

```
bool GetAddCarbonSkeletonWithBondStereo() const
```

Returns whether the carbon skeleton of a ring system retaining cis/trans bond stereo configuration is added to the dictionary. The default value is `true`.

See also

- *OE2DRingDictionaryCreatorOptions.SetAddCarbonSkeletonWithBondStereo* method

GetRetainExistingBuiltInTemplates

```
bool GetRetainExistingBuiltInTemplates() const
```

Sets whether a ring system will be added to the dictionary even if a corresponding layout already exists in the built-in ring dictionary.

See also

- *OE2DRingDictionaryCreatorOptions.SetRetainExistingBuiltInTemplates* method

GetRetainExistingTemplates

```
bool GetRetainExistingTemplates() const
```

Returns whether to override the ring template that already exists in the dictionary.

See also

- *OE2DRingDictionaryCreatorOptions.SetRetainExistingTemplates* method

SetAddCarbonSkeleton

```
void SetAddCarbonSkeleton(bool add)
```

Sets whether the carbon skeleton of a ring system is added to the dictionary.

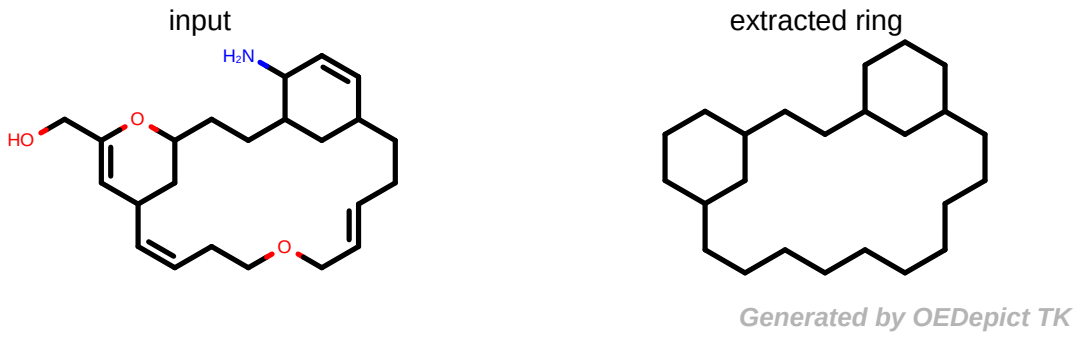


Fig. 1: **Example of extracting the carbon skeleton of a ring system**

SetAddCarbonSkeletonWithBondStereo

```
void SetAddCarbonSkeletonWithBondStereo(bool add)
```

Sets whether the carbon skeleton of a ring system retaining cis/trans bond stereo configuration is added to the dictionary. The default value is `true`.

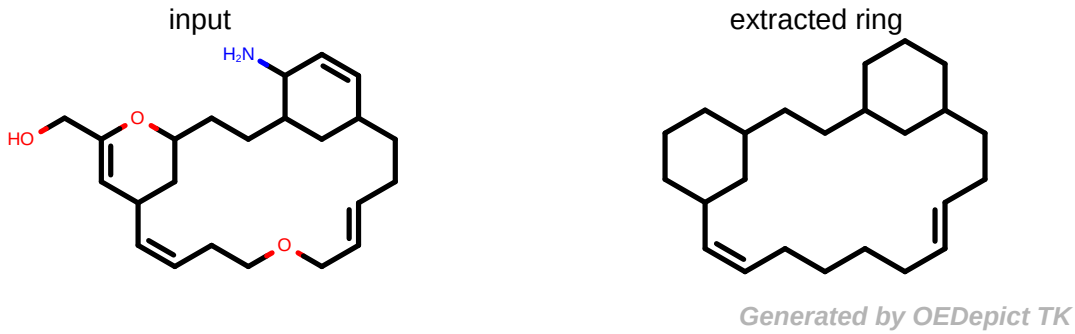


Fig. 2: **Example of extracting the carbon skeleton of a ring system with cis/trans bond stereo configuration**

SetRetainExistingBuiltInTemplates

```
void SetRetainExistingBuiltInTemplates(bool)
```

Sets whether a ring system will be added to the dictionary even if a corresponding layout already exists in the built-in ring dictionary. The default value is `false` i.e. user-defined ring layouts will have higher priority than the built-in ring templates.

SetRetainExistingTemplates

```
void SetRetainExistingTemplates(bool)
```

Sets whether to override the ring template that already exists in the dictionary. The default value is `false` i.e. the new user-defined ring template will be ignored.

OE3DMolStyle

```
class OE3DMolStyle
```

This class represents *OE3DMolStyle*.

The *OE3DMolStyle* class is the abstract interface for representing a 3D molecule style within *OEChem*.

Constructors

```
OE3DMolStyle()
OE3DMolStyle(const OE3DMolStyle& other)
```

Default and copy constructors.

ClearAtomColorer

```
void ClearAtomColorer()
```

Clears the value of the atom color scheme set to the OE3DMolStyle.

ClearAtomLabelColorer

```
void ClearAtomLabelColorer()
```

Clears the value of the color set for the atom label of the OE3DMolStyle.

ClearAtomLabelDataTag

```
void ClearAtomLabelDataTag()
```

Clears the value of the data tag set for the atom label of the OE3DMolStyle.

ClearAtomLabelText

```
void ClearAtomLabelText()
```

Clears the value of the text set for the atom label of the OE3DMolStyle.

ClearAtomLabelType

```
void ClearAtomLabelType()
```

Clears the value of the lable type set for the atoms of the OE3DMolStyle.

ClearAtomStyle

```
void ClearAtomStyle()
```

Clears the value of the atom type set for the OE3DMolStyle.

ClearContourStyle

```
void ClearContourStyle()
```

Clears the value of the contour type set for the OE3DMolStyle.

ClearGridContourColorer

```
void ClearGridContourColorer()
```

Clears the value of the contour type set for the OE3DMolStyle.

ClearGridType

```
void ClearGridType()
```

Clears the value of the grid type set for the OE3DMolStyle.

ClearHydrogenVisibility

```
void ClearHydrogenVisibility()
```

Clears the value of the hydrogen visibility set for the OE3DMolStyle.

ClearInteractionStyle

```
void ClearInteractionStyle()
```

Clears the value of the interaction style set for the OE3DMolStyle.

ClearProteinColorer

```
void ClearProteinColorer()
```

Clears the value of the color set for the proteins of the OE3DMolStyle.

ClearProteinStyle

```
void ClearProteinStyle()
```

Clears the value of the style set for the proteins of the OE3DMolStyle.

ClearSurfaceAlpha

```
void ClearSurfaceAlpha()
```

Clears the value of the surface alpha set for the OE3DMolStyle.

ClearSurfaceColorer

```
void ClearSurfaceColorer()
```

Clears the value of the color set for the surface of the OE3DMolStyle.

ClearSurfaceStyle

```
void ClearSurfaceStyle()
```

Clears the value of the cstyle set for the surface of the OE3DMolStyle.

ClearSurfaceType

```
void ClearSurfaceType()
```

Clears the value of the surface type for the OE3DMolStyle.

FromString

```
bool FromString(const std::string& str)
```

GetAtomColorer

```
const OEMolStyleColorer& GetAtomColorer() const
```

Returns an OEMolStyleColorer that specifies the color of the atoms set in the OE3DMolStyle.

GetAtomLabelColorer

```
const OEMolStyleColorer& GetAtomLabelColorer() const
```

Returns an OEMolStyleColorer that specifies the color of the atom label set in the OE3DMolStyle.

GetAtomLabelDataTag

```
std::string GetAtomLabelDataTag() const
```

Returns a string that specifies the data tag of the atom label set in the OE3DMolStyle.

GetAtomLabelPrecision

```
unsigned int GetAtomLabelPrecision() const
```

Returns an integer that correlates to the precision of the atom label set in the OE3DMolStyle.

GetAtomLabelText

```
std::string GetAtomLabelText() const
```

Returns a string of the atom label specified in the OE3DMolStyle.

GetAtomLabelType

```
unsigned int GetAtomLabelType() const
```

Returns an integer that specifies the atom label type set in the OE3DMolStyle.

GetAtomStyle

```
unsigned int GetAtomStyle() const
```

Returns an integer that specifies the atom style set in the OE3DMolStyle.

GetContourStyle

```
unsigned int GetContourStyle() const
```

Returns an integer that specifies the contour style set in the OE3DMolStyle.

GetGridContourColorer

```
const OEMolStyleColorer& GetGridContourColorer() const
```

Returns an OEMolStyleColorer that specifies the color scheme of the grid contour set in the OE3DMolStyle.

GetGridType

```
unsigned int GetGridType() const
```

Returns an integer that specifies the grid type set in the OE3DMolStyle.

GetHydrogenVisibility

```
unsigned int GetHydrogenVisibility() const
```

Returns an integer that specifies the hydrogen visibility set in the OE3DMolStyle.

GetInteractionStyle

```
unsigned int GetInteractionStyle() const
```

Returns an integer that specifies the interaction style set in the OE3DMolStyle.

GetProteinColorer

```
const OEMolStyleColorer& GetProteinColorer() const
```

Returns an OEMolStyleColorer that specifies the color scheme of the proteins set in the OE3DMolStyle.

GetProteinStyle

```
unsigned int GetProteinStyle() const
```

Returns an integer that specifies the protein style set in the OE3DMolStyle.

GetString

```
std::string GetString() const
```

GetSurfaceAlpha

```
unsigned int GetSurfaceAlpha() const
```

Returns an integer that specifies the surface alpha set in the OE3DMolStyle.

GetSurfaceColorer

```
const OEMolStyleColorer& GetSurfaceColorer() const
```

Returns an OEMolStyleColorer that specifies the color scheme of the surface set in the OE3DMolStyle.

GetSurfaceStyle

```
unsigned int GetSurfaceStyle() const
```

Returns an integer that specifies the surface style set in the OE3DMolStyle.

GetSurfaceType

```
unsigned int GetSurfaceType() const
```

Returns an integer that specifies the surface type set in the OE3DMolStyle.

HasAtomColorer

```
bool HasAtomColorer() const
```

Return a boolean indicated whether the OE3DMolStyle has a set atom colorer.

HasAtomLabelColorer

```
bool HasAtomLabelColorer() const
```

Return a boolean indicated whether the OE3DMolStyle has a set atom label colorer.

HasAtomLabelDataTag

```
bool HasAtomLabelDataTag() const
```

Return a boolean indicated whether the OE3DMolStyle has a set atom label data tag.

HasAtomLabelText

```
bool HasAtomLabelText() const
```

Return a boolean indicated whether the OE3DMolStyle has a set atom label text.

HasAtomLabelType

```
bool HasAtomLabelType() const
```

Return a boolean indicated whether the OE3DMolStyle has a set atom label type.

HasAtomStyle

```
bool HasAtomStyle() const
```

Return a boolean indicated whether the OE3DMolStyle has a set atom style.

HasContourStyle

```
bool HasContourStyle() const
```

Return a boolean indicated whether the OE3DMolStyle has a set contour style.

HasGridContourColorer

```
bool HasGridContourColorer() const
```

Return a boolean indicated whether the OE3DMolStyle has a set grid contour colorer.

HasGridType

```
bool HasGridType() const
```

Return a boolean indicated whether the OE3DMolStyle has a set grid type.

HasHydrogenVisibility

```
bool HasHydrogenVisibility() const
```

Return a boolean indicated whether the OE3DMolStyle has a set hydrogen visibility.

HasInteractionStyle

```
bool HasInteractionStyle() const
```

Return a boolean indicated whether the OE3DMolStyle has a set interaction style.

HasProteinColorer

```
bool HasProteinColorer() const
```

Return a boolean indicated whether the OE3DMolStyle has a set protein colorer.

HasProteinStyle

```
bool HasProteinStyle() const
```

Return a boolean indicated whether the OE3DMolStyle has a set protein style.

HasSurfaceAlpha

```
bool HasSurfaceAlpha() const
```

Return a boolean indicated whether the OE3DMolStyle has a set surface alpha.

HasSurfaceColorer

```
bool HasSurfaceColorer() const
```

Return a boolean indicated whether the OE3DMolStyle has a set surface colorer.

HasSurfaceStyle

```
bool HasSurfaceStyle() const
```

Return a boolean indicated whether the OE3DMolStyle has a set surfacestyle.

HasSurfaceType

```
bool HasSurfaceType() const
```

Return a boolean indicated whether the OE3DMolStyle has a set surface type.

IsValid

```
bool IsValid() const
```

Return a boolean indicated whether the OE3DMolStyle is valid or not.

SetAtomColorer

```
OE3DMolStyle& SetAtomColorer(const OEMolStyleColorer& colorer)
```

Sets the atom colorer for the OE3DMolStyle.

See also

- *OEMolStyleColorer*
- *OEAtomColorScheme*

SetAtomLabelColorer

```
OE3DMolStyle& SetAtomLabelColorer(const OEMolStyleColorer& colorer)
```

Sets the atom label colorer for the OE3DMolStyle.

See also

- *OEMolStyleColorer*

SetAtomLabelDataTag

```
OE3DMolStyle& SetAtomLabelDataTag(const std::string& tag,
                                  unsigned int precision = 0)
```

Sets the atom label data tage and percision for the OE3DMolStyle.

SetAtomLabelText

```
OE3DMolStyle& SetAtomLabelText(const std::string& label)
```

Sets the atom label text for the OE3DMolStyle.

SetAtomLabelType

```
OE3DMolStyle& SetAtomLabelType(unsigned int labelData,
                               unsigned int precision = 0)
```

Sets the atom label type and precision for the OE3DMolStyle.

SetAtomStyle

```
OE3DMolStyle& SetAtomStyle(unsigned int displayStyle)
```

Sets the atom style for the OE3DMolStyle.

See also

- *OEAtomStyle*

SetContourStyle

```
OE3DMolStyle& SetContourStyle(unsigned int displayStyle)
```

Sets the contour style for the OE3DMolStyle.

See also

- *OEContourStyle*

SetGridContourColorer

```
OE3DMolStyle& SetGridContourColorer(const OEMolStyleColorer& colorer)
```

Sets the grid contour colorer for the OE3DMolStyle.

See also

- *OEMolStyleColorer*

SetGridType

```
OE3DMolStyle& SetGridType(unsigned int gridType)
```

Sets the grid type for the OE3DMolStyle.

See also

- *OEStyleGridType*

SetHydrogenVisibility

```
OE3DMolStyle& SetHydrogenVisibility(unsigned int mode)
```

Sets the hydrogen visibility for the OE3DMolStyle.

See also

- *OEHydrogenVisibility*

SetInteractionStyle

```
OE3DMolStyle& SetInteractionStyle(unsigned int mode)
```

Sets the interaction style for the OE3DMolStyle.

See also

- *OEInteractionStyle*

SetProteinColorer

```
OE3DMolStyle& SetProteinColorer(const OEMolStyleColorer& colorer)
```

Sets the protein colorer for the OE3DMolStyle.

See also

- *OEMolStyleColorer*
- *OEAtomColorScheme*
- *OEProteinColorScheme*

SetProteinStyle

```
OE3DMolStyle& SetProteinStyle(unsigned int displayStyle)
```

Sets the protein style for the OE3DMolStyle.

See also

- *OEProteinStyle*

SetSurfaceAlpha

```
OE3DMolStyle& SetSurfaceAlpha(unsigned int alpha)
```

Sets the surface alpha for the OE3DMolStyle.

SetSurfaceColorer

```
OE3DMolStyle& SetSurfaceColorer(const OEMolStyleColorer& colorer)
```

Sets the surface colorer for the OE3DMolStyle.

See also

- *OEMolStyleColorer*
- *OESurfaceColorScheme*

SetSurfaceStyle

```
OE3DMolStyle& SetSurfaceStyle(unsigned int displayStyle)
```

Sets the surface style for the OE3DMolStyle.

See also

- *OESurfaceStyle*

SetSurfaceType

```
OE3DMolStyle& SetSurfaceType(unsigned int surfaceType)
```

Sets the surface type for the OE3DMolStyle.

See also

- *OESurfaceType*

See also

- *OEAddMolStyles* function
- *OEApplyStyleChange* function
- *OEClearStyle* function
- *OEGetAtomColor* function
- *OEGetAtomLabelColor* function
- *OEGetAtomLabelText* function
- *OEGetColorSchemeName* function
- *OEGetStyle* function
- *OEGetStyleChange* function
- *OEGetStyleIndex* function
- *OEHasStyle* function
- *OESetStyle* function
- *OESetStyleIndex* function
- *OESweepStyles* function

- *basicStyle* example
- *bFactorStyle* example
- *styleHierarchy* example

OEAbsCanonicalConfTest

```
class OEAbsCanonicalConfTest : public OEConfTestBase
```

This class represents *OEAbsCanonicalConfTest*.

This implementation of *OEConfTestBase* combines subsequent connection tables into a multi-conformer molecule if they have the same canonical SMILES, that is, absolute (nonisomeric) graph.

The following methods are publicly inherited from *OEConfTestBase*:

CombineMols	*CompareMols*	*CreateCopy*

CombineMols

```
bool CombineMols(OEMCMolBase& m1, OEMolBase& m2)
```

CompareMols

```
bool CompareMols(OEMolBase& m1, OEMolBase& m2) const
```

Returns whether the two *OEMolBase* object have the same canonical SMILES. Before the comparison all atom and bond stereo values of 'm2' are set to *OEAtomStereo.Undefined* and *OEBondStereo.Undefined*, respectively, followed by canonical ordering of its atoms by *OECanonicalOrderAtoms*

CreateCopy

```
OEConfTestBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAbsCanonicalConfTest* object is dynamically allocated and owned by the caller.

OEAbsoluteConfTest

```
class OEAbsoluteConfTest : public OEConfTestBase
```

This class represents *OEAbsoluteConfTest*.

This implementation of *OEConfTestBase* combines subsequent connection tables into a multi-conformer molecule if they:

1. Have the same number of atoms and bonds in the same order.
2. Each atom and bond must have identical properties with its order correspondent in the subsequent connection table.
3. Have the same title (optional).

> **Note**
>
> *OEAbsoluteConfTest* shouldn't be used for reading multi-conformer molecules in PDB or XYZ file format. These file formats typically require the use of *OEDetermineConnectivity* (and *OEPerceiveBondOrders*) which may potentially perceive and create bonds in different orders (or arbitrarily assign bond orders and Kekulé forms differently) for different conformers of the same molecule. This disrupts the ordering (and property) constraints described above. Generally, file formats other than PDB and XYZ should be used for storing multi-conformer molecules (as consecutive connection tables), but if unavoidable, the *OEAbsCanonicalConfTest* class should be used instead.

The following methods are publicly inherited from *OEConfTestBase*:

CombineMols	*CompareMols*	*CreateCopy*

Constructors

```
OEAbsoluteConfTest(bool compTitles = true)
```

Constructs an *OEAbsoluteConfTest* object. The argument 'compTitle' specifies whether to compare molecule titles (accessed by *OEMolBase.GetTitle*). If the constructor is called with no arguments or with the argument true, molecule titles will be required to be the same. Otherwise, the titles will not be compared. In the latter instance, each conformer will have the individual title of its original connection table and the multi-conformer molecule will reflect the title of the active conformer.

CombineMols

```
bool CombineMols(OEMCMolBase& m1, OEMolBase& m2)
```

CompareMols

```
bool CompareMols(OEMolBase& m1, OEMolBase& m2) const
```

Returns whether the two *OEMolBase* objects are the same by checking the three conditions listed at the top.

Before the comparison all atom and bond stereo values of 'm2' are set to *OEAtomStereo.Undefined* and *OEBondStereo.Undefined*, respectively, followed by canonical ordering of its atoms by *OECanonicalOrderAtoms*

CreateCopy

```
OEConfTestBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAbsoluteConfTest* object is dynamically allocated and owned by the caller.

OEAnisoUij

```
class OEAnisoUij
```

The *OEAnisoUij* class is a simple container used to package anisotropic B-factor parameters that can be associated with an *OEAtomBase* object. Using one can make accessing Uij values easier in wrapped languages.

See also

- *OEHasAnisou*
- *OEGetAnisou*
- *OESetAnisou*

Constructors

```
OEAnisoUij()
OEAnisoUij(int u11, int u22, int u33, int u12, int u13, int u23)
```

Default constructor (which sets all parameters to 0) and constructor that takes specific values for each parameter.

GetU11

```
int GetU11() const
```

GetU12

```
int GetU12() const
```

GetU13

```
int GetU13() const
```

GetU22

```
int GetU22() const
```

GetU23

```
int GetU23() const
```

GetU33

```
int GetU33() const
```

SetU11

```
void SetU11(int val)
```

SetU12

```
void SetU12(int val)
```

SetU13

```
void SetU13(int val)
```

SetU22

```
void SetU22(int val)
```

SetU23

```
void SetU23(int val)
```

SetU33

```
void SetU33(int val)
```

OEAtomBase

```
class OEAtomBase : public OESystem::OEBase
```

The *OEAtomBase* class is the abstract interface for representing atoms within **OpenEye's Toolkits**. An *OEAtomBase* object is created by calling the *NewAtom* method on the parent molecule.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEQAtomBase*

Assignment operator.

AddBond

```
bool AddBond(const OEBondBase *)=0
```

Clear

```
void Clear()=0
```

GetAtomicNum

```
unsigned int GetAtomicNum() const = 0
```

Returns the 'atomic number' property of the atom. A value of zero is returned for wildcard atoms, such as MDL super atoms. An atomic number is specified as the argument to *OEMolBase.NewAtom* and may be changed using *OEAtomBase.SetAtomicNum*. The set of valid values for this property are defined by the *OEElemNo* namespace.

GetAtoms

```
OESystem::OEIterBase<OEAtomBase>* GetAtoms() const = 0
```

Returns an iterator over the neighboring atoms of this atom. The returned iterator traverses only the explicit atoms that are bonded to the *OEAtomBase*. The number of neighbors in the iterator is identical to *OEAtomBase.GetExplicitDegree*.

```
OESystem::OEIterBase<OEAtomBase>* GetAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase> &) const = 0
```

Returns an iterator over the neighboring atoms of the *OEAtomBase* that match the predicate (*OEUnaryPredicate*)

GetAxial

```
bool GetAxial(const OEBondBase *) const
```

> **Warning**
>
> This is a deprecated API.

GetBond

```
OEBondBase* GetBond(const OEAtomBase *) const
```

Returns a *OEBondBase* pointer that connecting the *OEAtomBase* to the atom specified in the method argument. If the *OEAtomBase* is not connected to the specified atom, a NULL pointer (*OEBondBase**)0 is returned.

GetBonds

```
OESystem::OEIterBase<OEBondBase>* GetBonds() const = 0
```

Returns an iterator over the bonds connected to the atom. The returned iterator traverses only the explicit bonds that are attached to the *OEAtomBase*. The number of bonds in the iterator is identical to *OEAtomBase.GetExplicitDegree*.

```
OESystem::OEIterBase<OEBondBase>* GetBonds(const OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase> &) const = 0
```

Returns an iterator over the bonds of the *OEAtomBase* that match the predicate (*OEUnaryPredicate*)

GetDegree

```
unsigned int GetDegree() const
```

Returns the total number of neighbor atoms bonded to an atom, or equivalently the total number of bonds connected to an atom. This value includes bonds to implicit hydrogen atoms. This value is identical to *OEAtomBase.GetExplicitDegree()* + *OEAtomBase.GetImplicitHCount()*, and is also identical to *OEAtomBase.GetHvyDegree()* + *OEAtomBase.GetTotalHCount()*.

GetExplicitDegree

```
unsigned int GetExplicitDegree() const = 0
```

Returns the number of explicit neighbor atoms bonded to an atom. This value does not include bonds to implicit hydrogen atoms.

GetExplicitHCount

```
unsigned int GetExplicitHCount() const
```

Returns the number of explicit hydrogen neighbors bonded to an atom.

GetExplicitValence

```
unsigned int GetExplicitValence() const
```

Returns the sum of the bond orders of explicit bonds connected to an atom.

GetFormalCharge

```
int GetFormalCharge() const = 0
```

Returns the 'formal charge' property of an atom. The default value is zero. The 'formal charge' property may be set using the *SetFormalCharge* method. Formal charges may also be set on a molecule using the *OEAssignFormalCharges* function that infers reasonable formal charges using the atomic number, and implicit hydrogen count for each atom.

GetHvyDegree

```
unsigned int GetHvyDegree() const
```

Returns the number of non-hydrogen neighbors of an atom.

GetHvyValence

```
unsigned int GetHvyValence() const
```

Returns the sum of bond orders of bonds to non-hydrogen atoms.

GetHyb

```
unsigned int GetHyb() const = 0
```

Returns the 'hybridization' property of an atom. The set of valid values for this property are defined in the *OEHybridization* namespace. The default value is zero, (*OEHybridization.Unknown*). The 'hybridization' property can be set with the *OEAtomBase.SetHyb* method.

GetIdx

```
unsigned int GetIdx() const = 0
```

Returns the atom index of an atom. This value is assigned when an atom is created. The atom index is unique for the atoms of a given molecule, and is stable (not reused) for the lifetime of that *OEMolBase*. These values may not be assigned sequentially, and may contain gaps. Atom indices are guaranteed to be less than *OEMolBase.GetMaxAtomIdx*. Atom indices are typically only used for efficiently storing data in arrays (or vectors) externally to the *OEMolBase*. They arrays can be conveniently indexed via *OEAtomBase.GetIdx*. For iterating across the atoms of a molecule, use *OEMolBase.GetAtoms* and for keeping track of a particular atom use an *OEAtomBase**.

See also

- *Atom, Bond, and Conformer Indices* chapter

GetImplicitHCount

```
unsigned int GetImplicitHCount() const = 0
```

Returns the 'implicit hydrogen count' property of an atom. The default value is zero. The 'implicit hydrogen count' property may be set using the *OEAtomBase.SetImplicitHCount* method. Implicit hydrogen counts may also be set on a molecule using the *OEAssignImplicitHydrogens* function. The 'implicit hydrogen count' property of an atom is a stored, not a computed, property. To calculate a reasonable implicit hydrogen count for an atom from connectivity, use the *OEDefaultImplicitHCount* function.

To convert explicit hydrogens to implicit hydrogens, use the *OESuppressHydrogens* function, and to convert implicit hydrogens to explicit hydrogens, use the *OEAddExplicitHydrogens* function.

GetIntType

```
int GetIntType() const = 0
```

Returns the 'integer atom type' of an atom. The default value is zero. The 'integer atom type' property may be set using the *OEAtomBase.SetIntType* method.

GetIsotope

```
unsigned int GetIsotope() const = 0
```

Returns the 'isotopic mass' property of an atom. The default value is zero, meaning that the atom isn't a specific single isotope, but a composition of isotopes in their naturally occurring abundances. A non-zero value indicates that the atom is a specific isotope. The 'isotopic mass' property may be set using the *OEAtomBase.SetIsotope* method.

GetMapIdx

```
unsigned int GetMapIdx() const = 0
```

Returns the 'reaction map index' of property of an atom. This integer value is used for tracking equivalent atom positions in a reaction molecule/transform. The default value is zero, meaning that the atom isn't mapped in the reaction. This property is also used to store 'R-group' indices, for example when parsing the SMILES string `CCC[R1]`.

See also

- *OEAtomBase.SetMapIdx* method
- *OEIsRGroup* predicate

GetName

```
const char* GetName() const = 0
```

Returns the 'atom name' property of an atom. This property is typically used when reading or writing molecular file formats. For example in PDB files, the atom name `` CA `` is used to denote that the atom is a peptide alpha carbon. The default value is the empty string. The 'atom name' property may be set using the *OEAtomBase.SetName* method.

GetParent

```
OEMolBase* GetParent() const = 0
```

Returns the parent molecule (*OEMolBase*) of an atom. All *OEAtomBases* are created as a component of an *OEMolBase*, this method can be used to determine the molecule of which an atom is part. The 'parent molecule' property of an atom cannot be assigned, as its determined at the point the atom is created.

GetPartialCharge

```
double GetPartialCharge() const = 0
```

Returns the 'partial charge' property of an atom. The default value is `0.0`. The 'partial charge' property on an atom may be set using the *OEAtomBase.SetPartialCharge* method. The 'partial charge' property is a stored, not a calculated, property. The *OEClearPartialCharges* function resets the partial charges on a molecule to zero, and the *OEFormalPartialCharges* function sets the partial charge on each atom equal to its formal charge.

GetRadius

```
double GetRadius() const
```

Returns the 'radius' property of an atom. The default value is 0.0. The 'radius' property on an atom may be set using the *OEAtomBase.SetRadius* method. The 'radius' property is typically used to hold a Van der Waal's or ionic radius for a given atom when passing an *OEMolBase* to an algorithm requiring some form of radii, such as a surface area calculation.

GetRxnRole

```
unsigned int GetRxnRole() const
```

Returns the 'reaction role' property of an atom in a reaction molecule/transform. The set of valid values for this property are defined in the namespace *OERxnRole*. The default value is zero, *OERxnRole.None*, which means that the atom does have a role, *i.e.* is part of a normal molecule. Other values include *OERxnRole.Reactant*, *OERxnRole.Agent* and *OERxnRole.Product* which correspond to reactant, agent and product atoms, respectively, in a reaction or transform. The 'reaction role' property may be set using the *OEAtomBase.SetRxnRole* method.

GetStereo

```
unsigned int GetStereo(const std::vector<OEAtomBase *>& neighs,
                       unsigned int stereoclass) const
```

Returns the stereochemistry annotation of the atom.

neighs
: A vector of neighboring atoms and a specific stereochemistry class and returns whether that stereochemistry is specified at this atom, and if so, what that specified value is.

stereoclass
: **OEChem TK** currently only supports the tetrahedral atom stereochemistry class. The only valid *stereoclass* value is *OEAtomStereo.Tetra*.

This method returns the value *OEAtomStereo.Undefined* if the stereo chemistry for the given stereochemistry class is unspecified, if the stereo chemistry class is invalid or if the vector of neighboring atoms is invalid.

For a specified tetrahedral stereo center, it returns either *OEAtomStereo.Left* or *OEAtomStereo.Right*.

The stereochemistry annotation of an atom may be set using the *OEAtomBase.SetStereo* method.

See also

- stereo atom
- *Stereochemistry Perception* chapter
- *OEAtomBase.IsChiral*
- *OEAtomBase.HasStereoSpecified*

GetSymmetryClass

```
unsigned int GetSymmetryClass() const = 0
```

Returns the 'symmetry class' property of an atom. The default value is zero, indicating that symmetry classes haven't been assigned. The 'symmetry class' property may be set using the *OEAtomBase.SetSymmetryClass* method. Symmetry classes of a molecule are usually set by calling the *OEPerceiveSymmetry* function.

GetTotalHCount

```
unsigned int GetTotalHCount() const
```

Returns the total number of hydrogen atoms attached to an atom. This value includes the implicit hydrogen atoms. This value is identical to *OEAtomBase.GetExplicitHCount()* + *OEAtomBase.GetImplicitHCount()*.

GetType

```
const char* GetType() const = 0
```

Returns the 'atom type name' property of an atom. This value is typically used when reading and writing molecular formats. For example, in Sybyl `mol2` file, the atom type name `N.pl3` means that the atom should be treated as a three-valent planar nitrogen. The default value is the empty string. The 'atom type name' property may be set using the *OEAtomBase.SetType* method.

GetValence

```
unsigned int GetValence() const
```

Returns the sum of all bond orders to an atom. This value includes bonds to implicit hydrogen atoms. This value is equal to *OEAtomBase.GetExplicitValence()* + *OEAtomBase.GetImplicitHCount()*.

HasAlphaBetaUnsat

```
bool HasAlphaBetaUnsat(bool includePandS = true) const
```

Warning

This is a deprecated method. Use *OEHasAlphaBetaUnsat* predicate instead.

HasStereoSpecified

```
bool HasStereoSpecified(unsigned int v = OEAtomStereo::All) const
```

Determines whether the given atom has specified stereochemistry. The optional argument specifies the class of stereochemistry to check for. The default value, *OEAtomStereo.All*, checks whether any class of stereochemistry is specified for the current atom.

If an atom has unspecified stereochemistry, and this method returns `false`, the molecule is assumed to represent either a racemic mixture or an undetermined isomer of this compound. If this method returns `true`, this atom has an explicit stereochemistry which may be retrieved using the *OEAtomBase.GetStereo* method.

This method initially returns `false` for a newly created atom. The stereochemistry of an atom may be set by calling the *OEAtomBase.SetStereo* method with the appropriate stereochemistry class and any value other than *OEAtomStereo.Undefined*. The atom stereochemistry may subsequently be cleared by calling the same method, with the appropriate stereochemistry class, but specifying the value *OEAtomStereo.Undefined*.

See also

- stereo atom
- *Stereochemistry Perception* chapter
- *OEAtomBase.IsChiral*
- *OEAtomBase.GetStereo*

IsAromatic

```
bool IsAromatic() const = 0
```

Returns the 'aromatic' property of an atom. The default value is `false`. The 'aromatic' property of an atom can be set using the *OEAtomBase.SetAromatic* method. The 'aromatic' property of atoms and bonds in a molecule are typically set by calling either the *OEAssignAromaticFlags* function and cleared by the *OEClearAromaticFlags* function.

See also

- *Aromaticity Perception* chapter

IsCarbon

```
bool IsCarbon() const
```

Determines whether an atom is a carbon atom. This method tests whether the atom's atomic number is *6*, and is equivalent to *OEAtomBase.GetAtomicNum()* == *OEElemNo.C*.

IsChiral

```
bool IsChiral() const
```

Determines whether the current atom is a stereogenic center. If this method returns `true`, the atom is a chiral atom and the different left-handed and right-handed configurations about this atom represent two distinct isomers. If this method returns `false`, this atom is not a chiral stereocenter.

Note

This chiral atom property is independent of the stereo atom property that records whether a particular left-handed or right-handed configuration is specified at a particular atom.

See also

- chiral atom
- *Stereochemistry Perception* chapter
- *OEAtomBase.HasStereoSpecified*
- *OEAtomBase.GetStereo*

IsConnected

```
bool IsConnected(const OEAtomBase *) const
```

Determines whether the atom is bonded to the given atom. To obtain the actual bond linking the atom to its neighbor use the *OEAtomBase.GetBond* method. This method is equivalent to *OEAtomBase.GetBond()* != (*OEBondBase**)0.

IsDataType

```
bool IsDataType(const void *) const
```

IsDeleted

```
bool IsDeleted() const = 0
```

IsHalogen

```
bool IsHalogen() const
```

Determines whether an atom is a halogen atom. This method tests whether the atom is Fluorine, Chlorine, Bromine, Iodine or Astatine (atomic number *OEElemNo.F*, *OEElemNo.Cl*, *OEElemNo.Br*, *OEElemNo.I* or *OEElemNo.As*).

IsHydrogen

```
bool IsHydrogen() const
```

Determines whether an atom is a hydrogen atom. This method tests whether the atom's atomic number is 1, and is equivalent to *OEAtomBase.GetAtomicNum()* == *OEElemNo.H*.

IsInRing

```
bool IsInRing() const = 0
```

Returns the 'in ring' property of an atom. The default value is `false`. The 'in ring' property of an atom may be set using the *OEAtomBase.SetInRing* method. The 'in ring' properties of atoms and bonds are typically set by calling *OEFindRingAtomsAndBonds* function.

To determine whether an atom is a member of a specific ring/cycle size, use the *OEAtomIsInRingSize* function.

See also

- *Ring Perception* chapter

IsMetal

```
bool IsMetal() const
```

Determines whether an atom is a metal atom. This tests whether the atom number is not *OEElemNo.H*, *OEElemNo.He*, *OEElemNo.B*, *OEElemNo.C*, *OEElemNo.N*, *OEElemNo.O*, *OEElemNo.F*, *OEElemNo.Ne*, *OEElemNo.Si*, *OEElemNo.P*, *OEElemNo.S*, *OEElemNo.Cl*, *OEElemNo.Ar*, *OEElemNo.As*, *OEElemNo.Se*, *OEElemNo.Br*, *OEElemNo.Kr*, *OEElemNo.Te*, *OEElemNo.I*, *OEElemNo.Xe*, *OEElemNo.At* or *OEElemNo.Rn*.

IsNitrogen

```
bool IsNitrogen() const
```

Determines whether an atom is a nitrogen atom. This method tests whether the atom's atomic number is 7, and is equivalent to *OEAtomBase.GetAtomicNum()* == *OEElemNo.N*.

IsOxygen

```
bool IsOxygen() const
```

Determines whether an atom is an oxygen atom. This method tests whether the atom's atomic number is 8, and is equivalent to *OEAtomBase.GetAtomicNum()* == *OEElemNo.O*.

IsPhosphorus

```
bool IsPhosphorus() const
```

Determines whether an atom is a phosphorus atom. This method tests whether the atom's atomic number is 15, and is equivalent to *OEAtomBase.GetAtomicNum()* == *OEElemNo.P*.

IsPolar

```
bool IsPolar() const
```

Determines whether an atom is neither a carbon (*OEElemNo.C*) nor hydrogen (*OEElemNo.H*).

IsPolarHydrogen

```
bool IsPolarHydrogen() const
```

Determines whether an atom is a hydrogen attached to a polar atom. If the atom's atomic number is not one, *i.e.* *OEAtomBase.IsHydrogen()* == `false`, then this method returns `false` too. Otherwise, this method returns `true` if any of its neighbors return `true` for *OEAtomBase.IsPolar*.

IsSulfur

```
bool IsSulfur() const
```

Determines whether an atom is a sulfur (sulphur) atom. This method tests whether the atom's atomic number is 16, and is equivalent to *OEAtomBase.GetAtomicNum()* == *OEElemNo.S*.

OrderBonds

```
bool OrderBonds(const std::vector<OEBondBase *> &)=0
```

RemoveBond

```
bool RemoveBond(OEBondBase *)=0
```

SetAromatic

```
bool SetAromatic(bool)=0
```

Sets the 'aromatic' property of an atom. This value is `false` by default. The 'aromatic' property of an atom may be retrieved using the *OEAtomBase.IsAromatic* method. This method is normally only used by the *OEClearAromaticFlags* function and the *OEAssignAromaticFlags* function.

See also

- *Aromaticity Perception* chapter

SetAtomicNum

```
bool SetAtomicNum(unsigned int)=0
```

Sets the 'atomic number' property of an atom. The set of valid values for this property is specified by the *OEElemNo*. The default value is specified when an atom is created with *OEMolBase.NewAtom* method. The 'atomic number' property of an atom can be retrieved using the *OEAtomBase.GetAtomicNum* method.

SetAxial

```
bool SetAxial(const OEBondBase *, bool)
```

Warning

This is a deprecated method.

SetChiral

```
bool SetChiral(bool)
```

See also

- chiral atom
- *Stereochemistry Perception* chapter
- *OEAtomBase.SetStereo*

SetFormalCharge

```
bool SetFormalCharge(int)=0
```

Sets the 'formal charge' property of an atom. The default value is zero. The 'formal charge' property may be retrieved using the *OEAtomBase.GetFormalCharge* method.

SetHyb

```
bool SetHyb(unsigned int)=0
```

Sets the 'hybridization' property of an atom. The set of valid values for this property is specified by the *OEHybridization* namespace. The default value is zero (*OEHybridization.Unknown*). The 'hybridization' property may be retrieved using the *OEAtomBase.GetHyb* method.

See also

- *OEGetHybridization*
- *OEAssignHybridization*

SetImplicitHCount

```
bool SetImplicitHCount(unsigned int)=0
```

Sets the 'implicit hydrogen count' property of an atom. The default value is zero. The 'implicit hydrogen count' property may be retrieved using the *OEAtomBase.GetImplicitHCount* method.

SetInRing

```
bool SetInRing(bool)
```

Sets the 'in ring' property of an atom. This property is usually set by the *OEFindRingAtomsAndBonds* function. The default value is `false`. The 'in ring' property of an atom may be retrieved using the *OEAtomBase.IsInRing* method.

See also

- *Ring Perception* chapter

SetIntType

```
bool SetIntType(int)=0
```

Sets the 'integer atom type' property of an atom. The default value is zero. The 'integer atom type' property may be retrieved using the *OEAtomBase.GetIntType* method. Note that IntType is considered volatile and can be overwritten by different algorithms and also is not retained during I/O.

SetIsotope

```
bool SetIsotope(unsigned int)=0
```

Sets the 'isotopic mass' property of an atom (*i.e.* number of protons and neutrons). The default value is zero, meaning that the atom isn't a specific single isotope, but a composition of isotopes in their naturally occurring abundances. A non-zero value indicates that the atom is a specific isotope. The 'isotopic mass' property can be retrieved using the *OEAtomBase.GetIsotope* method.

SetMapIdx

```
bool SetMapIdx(unsigned int)=0
```

Sets the 'reaction map index' property of an atom. This integer value is used for tracking equivalent atom positions in a reaction molecule/transform. The default value is zero, meaning that the atom isn't mapped in the reaction.

See also

- *OEAtomBase.GetMapIdx* method

SetName

```
bool SetName(const char *)=0
bool SetName(const std::string &)=0
```

Sets the 'atom name' property of an atom. This property is typically used when reading or writing molecular file formats. For example in PDB files, the atom name `` CA `` is used to denote that the atom is a peptide alpha carbon. The default value is the empty string. The 'atom name' property may be retrieved using the *OEAtomBase.GetName* method.

SetPartialCharge

```
bool SetPartialCharge(float)=0
bool SetPartialCharge(double)=0
```

Sets the 'partial charge' property of an atom. The default value is `0.0`. The 'partial charge' property may be retrieved using the *OEAtomBase.GetPartialCharge* method.

SetRadius

```
bool SetRadius(double)
```

Sets the 'radius' property of an atom. The assigned radius may be retrieved using the *OEAtomBase.GetRadius* method. The 'radius' property is typically used to hold a Van der Waal's or ionic radius for a given atom when passing an *OEMolBase* to an algorithm requiring some form of radii, such as a surface area calculation.

See also

- *OEAssignCovalentRadii*
- *OEAssignDelphiRadii*
- *OEAssignBondiVdWRadii*
- *OEAssignPaulingVdWRadii*
- *OEAssignHonigIonicCavityRadii*
- *OEAssignZap9Radii*

SetRxnRole

```
bool SetRxnRole(unsigned int)
```

Sets the 'reaction role' property of an atom in a reaction molecule/transform. The set of valid values for this property is defined by the *OERxnRole* namespace. The default value is zero, *OERxnRole.None*, which means that the atom does have a role, *i.e.* is part of a normal molecule. Other values include *OERxnRole.Reactant*, *OERxnRole.Agent* and *OERxnRole.Product* which correspond to reactant, agent and product atoms, respectively, in a reaction or transform. The 'reaction role' property may be retrieved using the *OEAtomBase.SetRxnRole* method.

SetStereo

```
bool SetStereo(const std::vector<OEAtomBase *> & neighs,
               unsigned int stereoclass, unsigned int stereovalue)
```

Sets the stereochemistry annotation of the atom.

neighs
A vector of neighboring atoms that defines the stereochemistry.

stereoclass
OEChem TK currently only supports the tetrahedral atom stereochemistry class. The only valid *stereoclass* value is *OEAtomStereo.Tetra*.

stereovalue
This value can be either *OEAtomStereo.Left* or *OEAtomStereo.Right* when specifying a tetrahedral atom stereo center, or *OEAtomStereo.Undefined* when unspecifying *i.e.* deleting an atom stereo center.

This method returns `true` if the atom stereo was correctly set, and `false` if the stereochemistry class is invalid, the stereochemistry value is invalid for the stereochemistry class, or the vector of neighboring atoms is invalid. The stereo information of an atom may be retrieved using the *OEAtomBase.GetStereo*.

If set successfully, to a value other than *OEAtomStereo.Undefined*, future calls to *OEAtomBase.HasStereoSpecified* with the given stereochemistry class (or *OEAtomStereo.All*) will return `true`. If successfully set to the value *OEAtomStereo.Undefined*, future calls to *OEAtomBase.HasStereoSpecified* will return `false`.

See also

- stereo atom
- *Stereochemistry Perception* chapter
- *OEAtomBase.SetChiral*

SetSymmetryClass

```
bool SetSymmetryClass(unsigned int)=0
```

Sets the symmetry class of an atom. This routine is normally only called by the *OEPerceiveSymmetry* function. The default value is zero, indicating that the symmetry class hasn't be perceived. The symmetry class of an atom may be retrieved by using the *OEAtomBase.GetSymmetryClass* function.

SetType

```
bool SetType(const char *)=0
bool SetType(const std::string &)=0
```

Sets the 'atom type name' property of an atom. This value is typically used when reading and writing molecular formats. For example, in Sybyl `mol2` file, the atom type name `N.pl3` means that the atom should be treated as a three-valent planar nitrogen. The default value is the empty string. The 'atom type name' property may be retrieved using the *OEAtomBase.GetType* method.

Sweep

```
bool Sweep()=0
```

OEAtomBondSet

```
class OEAtomBondSet : public OESystem::OERoleSet
```

This class represents *OEAtomBondSet* that is a container that holds atom and bond pointers.

The following methods are publicly inherited from *OERoleSet*:

AddRole	*GetRoles*
ClearRoles	*HasRole*
DeleteRole	*NumRoles*

Constructors

```
OEAtomBondSet()
```

Default constructor that generates an empty container.

```
OEAtomBondSet(rhs: OEAtomBondSet, amap: OEAtomBase = (OEAtomBase, bmap: OEBondBase =␣
↪(OEBondBase) -> OEAtomBondSet
```

Copy constructor.

```
OEAtomBondSet(OESystem::OEIter<OEChem::OEAtomBase>& atoms)
OEAtomBondSet(OESystem::OEIter<OEChem::OEBondBase>& bonds)
OEAtomBondSet(OESystem::OEIterBase<OEChem::OEAtomBase>* atoms)
OEAtomBondSet(OESystem::OEIterBase<OEChem::OEBondBase>* bonds)
OEAtomBondSet(OESystem::OEIter<OEChem::OEAtomBase>& atoms,
              OESystem::OEIter<OEChem::OEBondBase>& bonds)
OEAtomBondSet(OESystem::OEIterBase<OEChem::OEAtomBase>* atoms,
              OESystem::OEIterBase<OEChem::OEBondBase>* bonds)
```

Constructors that generate an *OEAtomBondSet* container with the given atoms and bonds.

Add

```
bool Add(const OEMolBase& mol,
         const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& atompred)
bool Add(const OEMolBase& mol,
         const OESystem::OEUnaryPredicate<OEChem::OEBondBase>& bondpred)
bool Add(const OEMolBase& mol,
         const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& atompred,
         const OESystem::OEUnaryPredicate<OEChem::OEBondBase>& bondpred)
bool Add(const OEAtomBondSet& abset)
```

Adds the atom and bond objects to the container. Returns `false` if given a null pointer or if the atoms or bonds being added do not belong to the same molecule previous objects in the set belong to.

See also

- *OEAtomBondSet.GetParent* method

AddAtom

```
bool AddAtom(OEChem::OEAtomBase* atom)
```

Adds an atom to the container. Returns `false` if given a null pointer or if the atom being added does not belong to the same molecule previous atoms and bonds in the set belong to.

See also

- *OEAtomBondSet.GetParent* method

AddAtoms

```
bool AddAtoms(const std::vector<OEChem::OEAtomBase *>& atoms)
```

Adds all the atoms to the container. Returns `false` if given a null pointer or if any atom being added does not belong to the same molecule previous atoms and bonds in the set belong to.

See also

- *OEAtomBondSet.GetParent* method

AddBond

```
bool AddBond(OEChem::OEBondBase* bond)
```

Adds a bond to the container. Returns `false` if given a null pointer or if the bond being added does not belong to the same molecule previous atoms and bonds in the set belong to.

See also

- *OEAtomBondSet.GetParent* method

AddBonds

```
bool AddBonds(const std::vector<OEChem::OEBondBase *>& bonds)
```

Adds all the bonds to the container. Returns `false` if given a null pointer or if any bond being added does not belong to the same molecule previous atoms and bonds in the set belong to.

See also

- *OEAtomBondSet.GetParent* method

Clear

```
void Clear()
```

Removes all atoms and bonds from the *OEAtomBondSet* and clears the *OERoleSet*.

ClearAtoms

```
void ClearAtoms()
```

Removes all atoms from the *OEAtomBondSet*.

ClearAtomsAndBonds

```
void ClearAtomsAndBonds()
```

Removes all atoms and bonds from the *OEAtomBondSet* but retains the existing *OERoleSet*.

ClearBonds

```
void ClearBonds()
```

Removes all bonds from the *OEAtomBondSet*.

CreateCopy

```
OESystem::OEAtomBondSet* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned object derived from *OEAtomBondSet* is dynamically allocated and owned by the caller.

HasAtom

```
bool HasAtom(const OEChem::OEAtomBase* atom) const
```

Returns whether `atom` already belongs to the set.

HasBond

```
bool HasBond(const OEChem::OEBondBase* bond) const
```

Returns whether bond already belongs to the set.

IsEmpty

```
bool IsEmpty() const
```

Returns whether the set contains any atoms or bonds.

GetAtoms

```
OESystem::OEIterBase<OEChem::OEAtomBase>* GetAtoms() const
```

Returns an iterator over the atoms that are stored in the *OEAtomBondSet* object.

```
OESystem::OEIterBase<OEAtomBase>* GetAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase> &) const = 0
```

Returns an iterator over all of the atoms in the container that match the specified atom predicate (*OEUnaryPredicate*).

GetBonds

```
OESystem::OEIterBase<OEChem::OEBondBase>* GetBonds() const
```

Returns an iterator over the bonds that are stored in the *OEAtomBondSet* object.

```
OESystem::OEIterBase<OEBondBase>* GetBonds(const OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase> &) const = 0
```

Returns an iterator over all of the bonds in the container that match the specified bond predicate (*OEUnaryPredicate*).

GetParent

```
const OEMolBase* GetParent() const
```

Returns the parent molecule of a *OEAtomBondSet* object. This method can be used to determine the molecule of which atoms and bonds the container stores. The 'parent molecule' property of a *OEAtomBondSet* object cannot be altered, as it is determined at the point the container is created. The *OEAtomBondSet.GetParent* method returns a null pointer if the container is empty.

NumAtoms

```
unsigned int NumAtoms() const
```

Returns the number of atoms stored in the *OEAtomBondSet* object.

```
unsigned int NumAtoms(const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& atompred)␣
↪const
```

Returns the number of atoms stored in the *OEAtomBondSet* object matching the provided predicate.

NumBonds

```
unsigned int NumBonds() const
```

Returns the number of bonds stored in the *OEAtomBondSet* object.

```
unsigned int NumBonds(const OESystem::OEUnaryPredicate<OEChem::OEBondBase>& bondpred)␣
↪const
```

Returns the number of bonds stored in the *OEAtomBondSet* object matching the provided predicate.

Remove

```
bool Remove(const OEMolBase& mol,
            const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& atompred)
bool Remove(const OEMolBase& mol,
            const OESystem::OEUnaryPredicate<OEChem::OEBondBase>& bondpred)
bool Remove(const OEMolBase& mol,
            const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& atompred,
            const OESystem::OEUnaryPredicate<OEChem::OEBondBase>& bondpred)
bool Remove(const OEAtomBondSet& abset)
```

Removes the objects from the container. Returns `false` if given a null pointer or if any of the objects being removed do not belong to the same molecule previous objects in the set belong to.

RemoveAtom

```
bool RemoveAtom(OEChem::OEAtomBase* atom)
```

Removes an atom from the container. Returns `false` if given a null pointer or if the atom being removed does not belong to the same molecule previous atoms in the set belong to.

RemoveAtoms

```
bool RemoveAtoms(const std::vector<OEChem::OEAtomBase *>& atoms)
```

Removes atoms from the container. Returns `false` if given a null pointer or if any atoms being removed do not belong to the same molecule previous atoms in the set belong to.

RemoveBond

```
bool RemoveBond(OEChem::OEBondBase* bond)
```

Removes a bond from the container. Returns `false` if given a null pointer or if the bond being removed does not belong to the same molecule previous bonds in the set belong to.

RemoveBonds

```
bool RemoveBonds(const std::vector<OEChem::OEBondBase *>& bonds)
```

Removes bonds from the container. Returns `false` if given a null pointer or if any bonds being removed do not belong to the same molecule previous bonds in the set belong to.

Sweep

```
bool Sweep()
```

Removes deleted atoms and bonds from the container.

Translate

```
bool Translate(OEAtomBase** amap, OEBondBase** bmap)
```

Translates the pointers in the container based on the provided mapping between the atoms or bonds in the array arguments. This can be used to convert an existing *OEAtomBondSet* container to refer to a copy of the molecule made from a subset operations.

See also

OESubsetMol

OEAtomIsInAgent

```
class OEAtomIsInAgent : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEAtomIsInAgent* functor that identifies atoms of the catalysts or solvents of a reaction.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.GetRxnRole* method returns *OERxnRole.Agent* for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAtomIsInAgent* object is dynamically allocated and owned by the caller.

OEAtomIsInChain

```
struct OEAtomIsInChain : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEAtomIsInChain* functor that identifies chain atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

See also

- *OEAtomIsInRing* predicate

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsInRing* method returns `false` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAtomIsInChain* object is dynamically allocated and owned by the caller.

OEAtomIsInProduct

```
class OEAtomIsInProduct : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEAtomIsInProduct* functor that identifies atoms of the product molecule(s) of a reaction.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.GetRxnRole* method returns *OERxnRole.Product* for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAtomIsInProduct* object is dynamically allocated and owned by the caller.

OEAtomIsInReactant

```
class OEAtomIsInReactant : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEAtomIsInReactant* functor that identifies atoms of the reactant molecule(s) of a reaction.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.GetRxnRole* method returns *OERxnRole.Reactant* for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAtomIsInReactant* object is dynamically allocated and owned by the caller.

OEAtomIsInResidue

```
class OEAtomIsInResidue : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEAtomIsInResidue* functor that identifies atoms (*OEAtomBase*) with a given residue (*OEResidue*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEAtomIsInResidue(const OEResidue& r)
```

Constructs the functor with *OEResidue* as the internally specified residue.

```
OEAtomIsInResidue(const OEAtomIsInResidue& rhs)
```

Copy constructor.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the atom is in the residue specified as the construction argument. This is the same as the value returned by *OESameResidue* (r, *OEAtomGetResidue* (atom)), where 'r' is the functor's construction *OEResidue* argument.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAtomIsInResidue* object is dynamically allocated and owned by the caller.

OEAtomIsInRing

```
struct OEAtomIsInRing : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEAtomIsInRing* functor that identifies ring atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

See also

- *OEAtomIsInChain* predicate

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsInRing* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAtomIsInRing* object is dynamically allocated and owned by the caller.

See also

- Example program *RingSubset.cs*
- Example program *ExtractScaffold.cs*

OEBondBase

```
class OEBondBase : public OESystem::OEBase
```

The *OEBondBase* class is the abstract interface for representing bonds within **OpenEye's Toolkits**. An *OEBondBase* object is created by calling the *OEMolBase.NewBond* method on the parent molecule.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEQBondBase*

Clear

```
void Clear()=0
```

GetBgn

```
OEAtomBase* GetBgn() const = 0
```

Returns the 'begin' (or 'source') atom of a bond. The begin atom of a bond is defined either when the bond is created with *OEMolBase.NewBond*, or possibly by a call to the *OEBondBase.SetBgn* method. If the begin atom was specified as the NULL pointer, (*OEAtomBase**)0, when the bond was created, this method may return a NULL pointer.

See also

- *OEBondBase.GetEnd* method

GetBgnIdx

```
unsigned int GetBgnIdx() const
```

Returns the atom index of the 'begin' (or 'source') atom of a bond. This method is equivalent to *OEBondBase.GetBgn()*-> *GetIdx*.

See also

- *Atom, Bond, and Conformer Indices* chapter

GetEnd

```
OEAtomBase* GetEnd() const = 0
```

Returns the 'end' (or 'destination') atom of a bond. The end atom of a bond is defined either when the bond is created with *OEMolBase.NewBond*, or possibly by a call to the *OEBondBase.SetEnd* method. If the begin atom was specified as the NULL pointer, (*OEAtomBase**)0, when the bond was created, this method may return a NULL pointer.

GetEndIdx

```
unsigned int GetEndIdx() const
```

Returns the atom index of the 'end' (or 'destination') atom of a bond. This method is equivalent to *OEBondBase.GetEnd()*-> *GetIdx*.

See also

- *Atom, Bond, and Conformer Indices* chapter

GetIdx

```
unsigned int GetIdx() const = 0
```

Returns the bond index of a bond. This value is assigned when a bond is created. The bond index is unique for the bonds of a given molecule, and is stable (not reused) for the lifetime of that *OEMolBase*. These values may not be assigned sequentially, and may contain gaps. Bond indices are guaranteed to be less than *OEMolBase.GetMaxBondIdx*. Bond indices are typically only used for efficiently storing data in arrays (or vectors) externally to the *OEMolBase*. The arrays can be conveniently indexed via *OEBondBase.GetIdx*. For iterating across the bonds of a molecule, use *OEMolBase.GetBonds* and for keeping track of a particular bond use its *OEBondBase**.

See also

- *Atom, Bond, and Conformer Indices* chapter

GetIntType

```
int GetIntType() const = 0
```

Returns the 'integer bond type' property of a bond. The default value is zero. The 'integer bond type' property may be set using the *OEBondBase.SetIntType* method.

GetNbr

```
OEAtomBase* GetNbr(const OEAtomBase * nbr) const = 0
```

Returns the atom across the bond from the specified atom. If the 'nbr' argument represents the begin atom of the bond, *i.e.* *OEBondBase.GetBgn*, this method returns the end atom, *i.e.* *OEBondBase.GetEnd*, and if the 'nbr' argument is the end atom, this method returns the begin atom (*OEBondBase.GetBgn*). If passed an atom other than the begin or end atoms, this method returns a NULL pointer, (*OEAtomBase**)0. This method is often used for traversing the atoms of a molecule.

GetOrder

```
unsigned int GetOrder() const = 0
```

Returns the 'bond order' property of a bond. The 'bond order' property is one, for single bonds, two for double bonds, three for triple bonds and four for quadruple bonds. The default value is specified when a bond is created using *OEMolBase.NewBond*. The 'bond order' property of a bond may be set using the *OEBondBase.SetOrder* method.

GetParent

```
OEMolBase* GetParent() const = 0
```

Returns the parent molecule of a bond. All *OEBondBases* are created as a component of an *OEMolBase*, this method can be used to determine the molecule of which a bond is part. The 'parent molecule' property of a bond cannot be assigned, as its determined at the point the bond is created.

GetStereo

```
unsigned int GetStereo(const std::vector<OEAtomBase *>& neighs,
                       unsigned int stereoclass) const
```

Returns the stereochemistry annotation of the bond.

neighs
: A vector of adjacent atoms of the bond.

stereoclass
: **OEChem TK** currently only supports the cis/trans bond stereochemistry class. The only valid *stereoclass* value is *OEBondStereo.CisTrans*.

This method returns the value *OEBondStereo.Undefined* if the stereo chemistry for the given stereochemistry class is unspecified, if the stereo chemistry class is invalid or if the vector of adjacent atoms is invalid.

For a specified cis/trans stereo center, it returns either *OEBondStereo.Cis* or *OEBondStereo.Trans*.

The stereochemistry annotation of a bond may be set using the *OEBondBase.SetStereo* method.

See also

- stereo bond
- *Stereochemistry Perception* chapter
- *OEBondBase.IsChiral* method
- *OEBondBase.HasStereoSpecified* method

GetType

```
const char* GetType() const = 0
```

Returns the 'bond type name' property of a bond. This value is typically used when reading and writing molecular formats. For example, in Sybyl `mol2` file, the bond type name `am` means that the bond should be treated as an amide bond. The default value is the empty string. The 'bond type name' property may be set using the *OEBondBase.SetType* method.

HasStereoSpecified

```
bool HasStereoSpecified(unsigned int v = OEBondStereo::All) const
```

Determines whether the given bond has specified stereochemistry. The optional argument specifies the class of stereochemistry to check for. The default value, *OEBondStereo.All*, checks whether any class of stereochemistry is specified for the current bond.

If a bond has unspecified stereochemistry, and this method returns `false`, the molecule is assumed to represent either a racemic mixture or an undetermined isomer of this compound. If this method returns `true`, this bond has an explicit stereochemistry which may be retrieved using the *OEBondBase.GetStereo* method.

This method initially returns `false` for a newly created bond. The stereochemistry of a bond may be set by calling the *OEBondBase.SetStereo* method with the appropriate stereochemistry class, c, and any value other than *OEBondStereo.Undefined*. The bond stereochemistry may subsequently be cleared by calling the same method, with the appropriate stereochemistry class, but specifying the value *OEBondStereo.Undefined*.

See also

- stereo bond
- *Stereochemistry Perception* chapter
- *OEBondBase.IsChiral* method
- *OEBondBase.GetStereo* method

IsAromatic

```
bool IsAromatic() const = 0
```

Returns the 'aromatic' property of a bond. The default value is `false`. The 'aromatic' property of a bond can be set using the *OEBondBase.SetAromatic* method. The 'aromatic' property of bonds and atoms in a molecule are typically set by calling either the *OEAssignAromaticFlags* function, or the *OEClearAromaticFlags* function.

See also

- *Aromaticity Perception* chapter

IsChiral

```
bool IsChiral() const
```

Determines whether the current bond is a stereogenic center. If this method returns `true`, the bond is a chiral bond and the cis and trans forms of this bond represent two distinct isomers. If this method returns `false`, this bond is not a chiral stereocenter.

Note

This chiral bond property is independent of the stereo bond property that records whether a particular cis or trans configuration is specified at a particular bond.

See also

- chiral bond
- *Stereochemistry Perception* chapter
- *OEBondBase.GetStereo* method
- *OEBondBase.HasStereoSpecified* method

IsDataType

```
bool IsDataType(const void *) const
```

IsDeleted

```
bool IsDeleted() const = 0
```

IsInRing

```
bool IsInRing() const = 0
```

Returns the 'in ring' property of a bond. The default value is `false`. The 'in ring' property of a bond may be set using the *OEBondBase.SetInRing* method. The 'in ring' properties of bonds and atoms are typically set by calling the *OEFindRingAtomsAndBonds* function.

To determine whether a bond is a member of a specific ring/cycle size, use the *OEBondIsInRingSize* function.

See also

- *Ring Perception* chapter

IsRotor

```
bool IsRotor() const
```

Determines whether a bond is freely rotatable

Note

Since the 'rotatable' property is depends on the 'in ring' property. The *OEFindRingAtomsAndBonds* function must be called before accessing the rotatable bond property via *OEBondBase.IsRotor*.

See also

- rotatable bond
- OEGetRotatableBondCount function

SetAromatic

```
bool SetAromatic(bool)=0
```

Sets the 'aromatic' property of a bond. This value is `false` by default. The 'aromatic' property of a bond may be retrieved using the *OEBondBase.IsAromatic* method. This method is normally only used by the *OEClearAromaticFlags* function and *OEAssignAromaticFlags* function.

See also

- *Aromaticity Perception* chapter

SetBgn

```
bool SetBgn(OEAtomBase *)=0
```

Defines the 'begin' (or 'source') atom of a bond. Normally, the begin atom of a bond is defined when the bond is created with *OEMolBase.NewBond*. However, it is occasionally useful to defer this decision, by passing the NULL pointer, (*OEAtomBase**)0, as the begin atom to *OEMolBase.NewBond*. The begin atom should then later be specified by calling this method, *OEBondBase.SetBgn*, to resolve the begin atom before the molecule is used. Note that it is not possible to set/change the begin atom of a bond, once it has been defined. To achieve the same behavior call the *OEMolBase.DeleteBond* method, followed by *OEMolBase.NewBond*, to first destroy the original and replace it with a new bond.

SetChiral

```
bool SetChiral(bool)
```

See also

- chiral bond
- *Stereochemistry Perception* chapter
- *OEBondBase.SetStereo*

SetEnd

```
bool SetEnd(OEAtomBase *)=0
```

Defines the 'end' (or 'destination') atom of a bond. Normally, the end atom of a bond is defined when the bond is created with *OEMolBase.NewBond*. However, it is occasionally useful to defer this decision, by passing the NULL pointer, (*OEAtomBase**)0, as the end atom to *OEMolBase.NewBond*. The end atom should then later be specified by calling this method, *OEBondBase.SetEnd*, to resolve the end atom before the molecule is used. Note that it is not possible to set/change the end atom of a bond, once it has been defined. To achieve the same behavior call the *OEMolBase.DeleteBond* method, followed by *OEMolBase.NewBond*, to first destroy the original and replace it with a new bond.

SetInRing

```
bool SetInRing(bool)
```

Sets the 'in ring' property of a bond. This property is usually set by the *OEFindRingAtomsAndBonds* function. The default value is `false`. The 'in ring' property of a bond can be retrieved using the *OEBondBase.IsInRing* method.

See also

- *Ring Perception* chapter

SetIntType

```
bool SetIntType(int)=0
```

Sets the 'integer bond type' property of a bond. The default value is zero. The 'integer bond type' property may be retrieved using the *OEBondBase.GetIntType* method.

SetOrder

```
bool SetOrder(unsigned int)=0
```

Sets the 'bond order' property of a bond. The 'bond order' property is one, for single bonds, two for double bonds, three for triple bonds and four for quadruple bonds. The default value is specified when a bond is created using *OEMolBase.NewBond*. The 'bond order' property of a bond may be retrieved using the *OEBondBase.GetOrder* method.

SetStereo

```
bool SetStereo(const std::vector<OEAtomBase *> & neighs,
               unsigned int stereoclass, unsigned int stereovalue)
```

Sets the stereochemistry annotation of the bond.

neighs
: A vector of neighboring atoms that defines the stereochemistry.

stereoclass
: **OEChem TK** currently only supports the cis/trans bond stereochemistry class. The only valid *stereoclass* value is *OEBondStereo.CisTrans*.

stereovalue
: This value can be either *OEBondStereo.Cis* or *OEBondStereo.Trans* when specifying a cis/trans bond stereo center, or *OEBondStereo.Undefined* when un-specifying *i.e.* deleting a bond stereo center.

This method returns `true` if the bond chirality was correctly set, and `false` if the stereochemistry class is invalid, the stereochemistry value is invalid for the stereochemistry class, or the vector of adjacent atoms is invalid. The stereo information of a bond may be retrieved using the *OEBondBase.GetStereo*.

If set successfully, to a value other than *OEBondStereo.Undefined*, future calls to *OEBondBase.HasStereoSpecified* with the given stereochemistry class (or *OEBondStereo.All*) will return `true`. If successfully set to the value *OEBondStereo.Undefined*, future calls to *OEBondBase.HasStereoSpecified* will return `false`.

See also

- stereo bond
- *Stereochemistry Perception* chapter
- *OEBondBase.SetChiral*

SetType

```
bool SetType(const char *)=0
bool SetType(const std::string &)=0
```

Sets the 'bond type name' property of a bond. This value is typically used when reading and writing molecular formats. For example, in Sybyl `mol2` file, the bond type name `am` means that the bond should be treated as an amide bond. The default value is the empty string. The 'bond type name' property may be retrieved using the *OEBondBase.GetType* method.

SwapEnds

```
bool SwapEnds()=0
```

Swaps the begin and end atoms of a bond. The lists of bonds contained in each of the two atoms are not altered. After calling the *OEBondBase.SwapEnds* method, the atoms returned by the *OEBondBase.GetBgn* and *OEBondBase.GetEnd* methods will be interchanged.

Sweep

```
bool Sweep()=0
```

OEBondIsInChain

```
struct OEBondIsInChain : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEBondIsInChain* functor that identifies chain bonds (*OEBondBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

See also

- *OEBondIsInRing* predicate

operator()

```
bool operator()(const OEBondBase& bond) const
```

Returns `true`, if the *OEBondBase.IsInRing* method returns `false` for the given *OEBondBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEBondIsInChain* object is dynamically allocated and owned by the caller.

OEBondIsInRing

```
struct OEBondIsInRing : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEBondIsInRing* functor that identifies ring bonds (*OEBondBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

See also

- *OEBondIsInChain* predicate

operator()

```
bool operator()(const OEBondBase& bond) const
```

Returns `true`, if the *OEBondBase.IsInRing* method returns `true` for the given *OEBondBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEBondIsInRing* object is dynamically allocated and owned by the caller.

OECIFCategory

```
struct OECIFCategory
```

This class represents the *OECIFCategory* that holds CIF header data and acts as an interface to interact with this header data. This object reflects what is contained in a CIF category.

The *CIF File Format* section explains in more detail the CIF data structure.

See also

- *OECIFData* class
- *OECIFOptions* class
- *CIF File Format*

Constructors

```
OECIFCategory()
OECIFCategory(const std::string& categoryName,
              const std::vector<std::string>& attributes,
              const std::vector<std::string>& values)
OECIFCategory(const std::string& categoryName,
              const std::string& attribute,
              const std::string& value)
OECIFCategory(const std::string& categoryName,
              const std::vector<std::string>& attributes)
OECIFCategory(const std::string& categoryName)
```

Create a *OECIFCategory* object.

AddAttribute

```
bool AddAttribute(const std::string& newAttribute,
                  const std::string& newValue = "?")
bool AddAttribute(const std::string& newAttribute,
                  const std::vector<std::string>& newValues)
```

Adds a column of values to the indicated CIF attribute. If no values are input, a question mark "?" value will be used to populate attribute values.

AddRow

```
bool AddRow(const std::vector<std::string>& nvals)
bool AddRow(const std::string init = "?")
```

Adds a row of value(s) to the CIF category.

If no values are input, by default, it will add a question mark "?" to every attribute in the category.

DeleteRow

```
bool DeleteRow(unsigned int row)
```

Deletes a category's indicated row.

DeleteAttribute

```
bool DeleteAttribute(const std::string& attribute)
```

Deletes a category attribute.

GetAttributes

```
std::vector<std::string> GetAttributes() const
```

Returns the category's attributes.

GetAttributeValue

```
std::string GetAttributeValue(const std::string& attribute,
                              bool raw = false) const
std::string GetAttributeValue(const std::string& attribute,
                              unsigned int row, bool raw = false) const
```

Gets the saved value on the indicated attribute. If multiple values exist for a given attribute, then a row number must be identified.

GetAttributeValues

```
std::vector<std::string> GetAttributeValues(const std::string& attribute,
                                            bool raw = false) const
```

Gets all the saved values on the indicated attribute.

GetAttributeIndex

```
int GetAttributeIndex(const std::string& attribute) const
```

Returns the attribute's base-0 index. For any given row of data associated with the category, this index correlates with the attribute's value in that row. For example, if GetAttributeIndex("type")=1, for every row generated by GetRows(), the _category.type data would be the [1] value in that row.

GetCategoryName

```
const std::string& GetCategoryName() const
```

Returns the category's name.

GetDataItem

```
std::string GetDataItem(const std::string& attribute) const
```

Returns the CIF data item name in the format: '_category.attribute'

GetDataItems

```
std::vector<std::string> GetDataItems() const
```

Returns all the CIF data item names in the category in the format: '_category.attribute'

GetNumRows

```
unsigned int GetNumRows() const
```

Number of CIF rows in the category.

GetNumAttributes

```
unsigned int GetNumAttributes() const
```

Number of CIF attributes in the category.

GetNumericAttributeValue

Gets the saved value for the indicated attribute and converts it to a numeric value. If multiple values exist for a given attribute, then a row number must be identified. If a value cannot be easily converted to a numeric value, it will default to 0.

GetNumericAttributeValues

Gets all the saved values for the indicated attribute and converts to a numeric value. If a value cannot be easily converted to a numeric value, it will default to 0.

GetRow

```
std::vector<std::string> GetRow(unsigned int row, bool raw = false) const
```

Returns as a specific row from the category. Row indicators are base-0. Raw values are formatted as they will appear in the header. This is most noticeable with values that contain space characters, as CIF formatting requires single or double quotes around such a string value.

GetRows

```
std::vector<std::vector<std::string>> GetRows(int beginRow,
                                              int endRow,
                                              bool raw = false) const
std::vector<std::vector<std::string>> GetRows(bool raw = false) const
```

Returns rows from the category. Row indicators are base-0. Raw values are formatted as they will appear in the header. This is most noticeable with values that contain space characters, as CIF formatting requires single or double quotes around such a string value.

GetRowIndices

```
std::vector<unsigned int> GetRowIndices(const std::string& attribute,
                                        const std::string& matchValue) const
```

Returns the base-0 row indices whose values string-match with input match values.

HasAttribute

```
bool HasAttribute(const std::string& attribute) const
```

Returns if an attribute is present in this object.

SetData

```
bool SetData(const std::string& attribute,
             const std::string& newValue)
bool SetData(const std::string& attribute, unsigned int row,
             const std::string& newValue,
             const std::string init = "?")
bool SetData(const std::string& attribute,
             const std::vector<std::string>& newValues)
bool SetData(unsigned int row,
             const std::vector<std::string>& newValues,
             const std::string init = "?")
```

Sets attribute and row data. When setting data related to row values, if the indicated row value exceeds the maximum number of rows, then empty values will populate any undefined attributes values. Rows indicators are base-0.

SortByAttribute

```
void SortByAttribute(const std::string& attribute)
```

All rows will be rearranged according to the alphanumeric order of the indicated attribute's values.

OECIFData

```
struct OECIFData
```

This class represents *OECIFData* that holds CIF header data and acts as an interface to interact with this header data. This object reflects what is contained in the CIF `data_` section.

The goal for the OECIFData class is to allow a user to freely interact with CIF data and be confident the data will be written in proper format that can be easily be read again. This object allows structural data stored on the molecule to be synchronized and updated with its CIF header data to reflect changes in the molecule. No change in OECIFData will affect the molecule that stores it.

The overall structure of OECIFData reflects CIF formatting. The OECIFData object stores all the data in a `data_` block. Values from each CIF category are stored on the OECIFCategory object. Thus, OECIFData is a collection of OECIFCategory objects mapped to the category name.

Access to the data using OECIFData utilizes the CIF category and attribute names. These functions allow a user to get, set or add any value in the CIF header.

Note

Category names must begin with an underscore and end with a period. Example: `"_entity."`

The *CIF File Format* section explains the CIF data structure in more detail.

See also

- *OEUpdateMMCIFData* function
- *OECIFCategory* class
- *OECIFOptions* class
- *CIF File Format*

Constructors

```
OECIFData()
OECIFData(const std::string& cifHeaderString)
OECIFData(const OEMolBase& mol)
```

Create an *OECIFData* object from a header data string, or header data stored on a mol accessed using *OEGetMMCIFData*.

AddCategory

```
bool AddCategory(const OECIFCategory& category)
bool AddCategory(const std::string& categoryName,
                 const std::vector<std::string>& attributes,
                 const std::vector<std::string>& newValues)
bool AddCategory(const std::string& categoryName,
                 const std::vector<std::string>& attributes)
bool AddCategory(const std::string& categoryName,
                 const std::string& attribute,
                 const std::vector<std::string>& newValues)
bool AddCategory(const std::string& categoryName,
                 const std::string& attribute,
                 const std::string& newValue)
bool AddCategory(const std::string& categoryName)
```

Adds a new category to the OECIFData object.

AddRow

```
bool AddRow(const std::string& categoryName,
            const std::vector<std::string>& newValues)
bool AddRow(const std::string& categoryName,
            const std::string init = "?")
```

Adds a row of values to the indicated CIF item prefix. If no values are input, by default, it will add a question mark "?" to every attribute in the category.

AddData

```
bool AddData(const std::string& categoryName,
             const std::string& attribute,
             const std::vector<std::string>& newValues)
bool AddData(const std::string& categoryName,
             const std::string& attribute,
             const std::string& newValue = "?")
```

Adds a new attribute and a column of its values to the indicated CIF item tag. The number of new data input must match the number of rows currently in the OECIFCategory it is being added to.

ChemCompToMol

```
bool ChemCompToMol(OEMolBase& cifMol, const std::string resName) const
```

Creates a molecule from CIF _chem_comp data in the OECIFData object. The following CIF categories must be present for proper function:

- _chem_comp_atom. defines the atom name and element
- _chem_comp_bond. defines the bonding network

The input cifMol molecule is cleared before parsing, so the resulting molecule object only contains atoms and bonds from the _chem_comp header data.

DeleteCategory

```
bool DeleteCategory(const std::string& categoryName)
```

Deletes a category based on the category name.

DeleteRow

```
bool DeleteRow(const std::string& categoryName, unsigned int row)
```

Deletes a category's indicated row values based on the category name.

DeleteAttribute

```
bool DeleteAttribute(const std::string& categoryName,
                     const std::string& attribute)
```

Deletes a category's attribute and associated values.

GenerateHeader

```
std::string GenerateHeader() const
```

Using all the categories saved on this object, this will output the CIF header.

GeneratePolySeqScheme

```
bool GeneratePolySeqScheme(OEMolBase& mol,
                           const unsigned int maxMismatch = 10)
```

Will generate the _pdbx_poly_seq_scheme header section using header and molecule data. The following categories need to be present and properly populated for this function:

- _struct_asym.
- _entity.
- _entity_poly_seq.

The molecule object must contain at least one residue. Any existing _pdbx_poly_seq_scheme data will be cleared before generation.

GetAttributes

```
std::vector<std::string>  GetAttributes(const std::string& categoryName) const
```

Gets a list of all the attributes in this object.

GetAttributeValue

```
std::string GetAttributeValue(const std::string& categoryName,
                              const std::string& attribute,
                              bool raw = false) const
std::string GetAttributeValue(const std::string& categoryName,
                              const std::string& attribute,
                              unsigned int row, bool raw = false) const
```

Gets the saved value on the indicated category's attribute. If multiple values exist for a given attribute, then a row number must be identified.

GetAttributeValues

```
std::vector<std::string> GetAttributeValues(const std::string& categoryName,
                                            const std::string& attribute,
                                            bool raw = false) const
```

Gets all the saved values on the indicated category's attribute.

GetAttributeIndex

```
int GetAttributeIndex(const std::string& categoryName,
                      const std::string& attribute) const
```

Returns the attribute's base-0 index. For any given row of data associated with the category, this index correlates with the attribute's value in that row. For example, if GetAttributeIndex("entity.", "type")=1, for every row generated by GetRows("_entity."), the _category.type data would be the [1] value in that row.

GetCategory

```
OECIFCategory& GetCategory(const std::string& categoryName)
```

Gets the *OECIFCategory* object by its category name.

GetCategoryNames

```
std::vector<std::string> GetCategoryNames() const
```

Returns all the CIF category names present in this object.

GetDataItems

```
std::vector<std::string>  GetDataItems(const std::string& categoryName) const
```

Returns a list of all the CIF data items in this object.

GetName

```
const std::string& GetName() const
```

Returns the name of the header data. Value is defined using the molecule's title.

GetNumCategories

```
unsigned int GetNumCategories() const
```

Number of CIF categories saved in this object.

GetNumCategoryRows

```
unsigned int GetNumCategoryRows(const std::string& categoryName) const
```

Number of CIF rows in a category.

GetNumCategoryAttributes

```
unsigned int GetNumCategoryAttributes(const std::string& categoryName) const
```

Number of CIF attributes in a category.

GetNumericAttributeValue

Gets the saved value on the indicated category's attribute and converts to a numeric value. If multiple values exist for a given item attribute, then a row number must be identified. If a value cannot be easily converted to a numeric value, it will default to 0.

GetNumericAttributeValues

Gets all the saved values on the indicated category's attribute and converts to a numeric value. If a value cannot be easily converted to a numeric value, it will default to 0.

GetRow

```
std::vector<std::string> GetRow(const std::string& categoryName,
                                unsigned int row, bool raw = false) const
```

Returns a specific row from the indicated category. Rows values are base-0. Raw values are formatted as they will appear in the header. This is most noticeable with values that contain space characters as CIF formatting requires single or double quotes around such a string value.

GetRows

```
std::vector<std::vector<std::string>> GetRows(const std::string& categoryName,
                                              int beginRow,
                                              int endRow,
                                              bool raw = false) const
std::vector<std::vector<std::string>> GetRows(const std::string& categoryName,
                                              bool raw = false) const
```

Returns rows from the indicated category. Rows values are base-0. Raw values are formatted as they will appear in the header. This is most noticeable with values that contain space characters as CIF formatting requires single or double quotes around such a string value.

GetRowIndices

```
std::vector<unsigned int> GetRowIndices(const std::string& categoryName,
                                        const std::string& attribute,
                                        const std::string& matchValue) const
```

Returns the base-0 row indices whose values string-match with input match value.

HasAttribute

```
bool HasAttribute(const std::string& categoryName,
                  const std::string& attribute) const
```

Returns if an attribute is present in this object.

HasCategoryName

```
bool HasCategoryName(const std::string& categoryName) const
```

Returns if a category name is present in this object.

SetCategory

```
bool SetCategory(const OECIFCategory& category)
bool SetCategory(const std::string& categoryName,
                 const std::string& attribute,
                 const std::string& val)
bool SetCategory(const std::string& categoryName,
                 const std::vector<std::string>& attributes,
                 const std::vector<std::string>& vals)
```

Sets OECIFCategory information in this object.

SetData

```
bool SetData(const std::string& categoryName,
             const std::string& attribute, const std::string& nval)
bool SetData(const std::string& categoryName,
             const std::string& attribute, unsigned int row,
             const std::string& nval, const std::string init = "?")
bool SetData(const std::string& categoryName,
             const std::string& attribute,
             const std::vector<std::string>& nvals)
bool SetData(const std::string& categoryName, unsigned int row,
             const std::vector<std::string>& nvals,
             const std::string init = "?")
```

Sets attribute and row data on the indicated category.

SetMMCIFChemCompData

```
bool SetMMCIFChemCompData(const OEMolBase& mol,
                          const OEMolBase& chemCompMol,
                          const bool strict = false)
```

Updates _chem_comp sections in the mmCIF header metadata with information from the input chemCompMol molecule. First, a single-residue chemCompMol is checked to name match with a residue in the molecule. The chemCompMol atom and bond data is then used to replace the _chem_comp header data for a similarly named residue in the mol. If strict is used, an additional check for atom name matching is made between the matched residue in the molecule and the chemCompMol.

The following categories need to be present and properly populated in the input molecules header for this function. If any attributes are missing, running OEUpdateMMCIFData with ChemComp perception should populate any missing attributes:

- _chem_comp.
- _chem_comp_atom.
- _chem_comp_bond.

If the molecule already exists, it will replace all chemical component reference of that molecule with the new molecule. By default, residue atom name matching is enforced. If strict is turned off, an *OESubSearch* is used to identify structural similarity with the new molecule. If a match is found, the matching structure atom names will be used in the _chem_comp atom naming scheme.

Update

```
bool Update(OEMolBase& mol, const OECIFOptions& opts)
```

Coordinates the molecule's structural and residue data with related information found in the header. Where there are disagreements, the molecule's data overrides the header data. Because updating can affect many parts of the header, to control what parts are and are not updated, the *OECIFOptions* class indicates which parts of the header will be perceived during the update.

This function can be used to convert PDB header data into CIF header data. Create an OECIFData object using the PDB molecule and run Update(). Only perceivable fields in the molecule's structure data will be populated.

ValidateMMCIFHeader

```
bool ValidateMMCIFHeader(bool allowErrors = false,
                         bool strict = false)
```

Validates self-consistency within a mmCIF header. Many items will reference each other and this function runs checks to see if some of these are consistent with each other. Item entries that are checked include:

- _entity.
- _entity_poly.
- _entity_poly_seq.
- _chem_comp.
- _struct_asym.

The strict flag will explore more parts of the header and enforce a wider range of requirements:

- Any 'entry_id' value reference must match an already defined '_entry.id'
- Any 'entity_id' value reference must match an already defined '_entity.id'
- An expression system has been identified (_entity_src_gen., _entity_src_nat., _entity_src_syn.)
- _chem_comp_atom.
- _chem_comp_bond.
- _struct_ref.
- _struct_ref_seq.

ValidateMMCIFMol

```
bool ValidateMMCIFMol(const OEMolBase& mol,
                      bool allowErrors = false, bool strict = false)
```

Validates self-consistency between the header and mol reference. Item entries that are checked against the input molecule include:

- _entity.
- _chem_comp.
- _struct_asym.

The strict flag will explore more parts of the header and enforce a wider range of requirements against the input molecule:

- strict _entity. header validation
- _pdbx_poly_seq_scheme.

OECIFOptions

```
class OECIFOptions
```

An option handling class for the *OEUpdateMMCIFData* function

and *OECIFData.Update* method in the *OECIFData* class. The *CIF File Format* section explains in more detail the perception process and what CIF attributes are affected when an OECIFOption is set to `true` for a given perception method.

See also

- *OECIFData* class
- *CIF File Format* section

Constructors

```
OECIFOptions()
OECIFOptions(const OECIFOptions &)
```

Default and copy a constructor that constructs an OECIFOptions object.

GetPerceiveEntity

```
bool GetPerceiveEntity() const
```

A value of `true` will perceive all entity-related CIF attributes.

GetPerceiveChemComp

```
bool GetPerceiveChemComp() const
```

A value of `true` will perceive all chemical component-related CIF attributes.

GetPerceiveCell

```
bool GetPerceiveCell() const
```

A value of `true` will perceive CIF attributes related to the structure cell.

GetPerceiveSymmetry

```
bool GetPerceiveSymmetry() const
```

A value of `true` will perceive CIF attributes related to structural symmetry.

GetPerceiveStruct

```
bool GetPerceiveStruct() const
```

A value of `true` will perceive all struct or structure-related CIF attributes.

GetPerceiveExperimental

```
bool GetPerceiveExperimental() const
```

A value of `true` will coordinate the experimental CIF attributes with the perceived header values.

GetPerceiveOther

```
bool GetPerceiveOther() const
```

All remaining CIF categories not covered in the other perception options are coordinated with the perceived header values.

SetPerceiveEntity

```
bool SetPerceiveEntity(const bool)
```

Sets whether to perceive entity-related CIF attributes during header updates when molecule data informs header values. Default is `false`.

SetPerceiveChemComp

```
bool SetPerceiveChemComp(const bool)
```

Sets whether to perceive chemical component-related CIF attributes during header updates when molecule data informs header values. Default is `false`.

SetPerceiveCell

```
bool SetPerceiveCell(const bool)
```

Sets whether to perceive CIF attributes related to the structure cell during header updates when molecule data informs header values. Default is `false`.

SetPerceiveSymmetry

```
bool SetPerceiveSymmetry(const bool)
```

Sets whether to perceive CIF attributes related to the structural symmetry during header updates when molecule data informs header values. Default is `false`.

SetPerceiveStruct

```
bool SetPerceiveEntity(const bool)
```

Sets whether to perceive struct or structure related CIF attributes during header updates when molecule data informs header values. Default is `false`.

SetPerceiveExperimental

```
bool SetPerceiveExperimental(const bool)
```

Sets whether to coordinate experimental CIF attributes with the perceived header values. Default is `false`.

SetPerceiveOther

```
bool SetPerceiveOther(const bool)
```

Sets whether to coordinate all remaining CIF categories not covered in the other perception options. Default is `false`.

OECliqueSearch

```
class OECliqueSearch
```

This class represents *OECliqueSearch*.

The *OECliqueSearch* class performs bounded common substructure searches. It is a useful search method in cases where common substructures other than the maximum common substructure (*OEMCSSearch*) need to be identified.

See also

- *Clique Search* section
- Example program *CliqueAlign.cs*

Constructors

```
OECliqueSearch(unsigned int type = OEMCSType::Default)
```

Default constructor.

```
OECliqueSearch(const OECliqueSearch &)
```

Copy constructor.

```
OECliqueSearch(const char* smarts,
               unsigned int type = OEMCSType::Default)
```

Constructs an *OECliqueSearch* object using a 'smarts' pattern. The SMARTS pattern is parsed into a query molecule (*OEQMolBase*). The valid types of the *OECliqueSearch* are defined in the *OEMCSType* namespace.

```
OECliqueSearch(const OEQMolBase &, bool perception = true,
               unsigned int type = OEMCSType::Default)
```

Constructs an *OECliqueSearch* object using a query molecule (*OEQMolBase*).

qmol
: The query molecule of the MCS search.

type
: The type of the *OECliqueSearch* from the *OEMCSType* namespace.

perception
If `true`, then ring perception is performed on the query molecule.

```
OECliqueSearch(const OEMolBase &, unsigned int atomexpr,
               unsigned int bondexpr,
               unsigned int type = OEMCSType::Default)
```

Constructs an *OECliqueSearch* object using a molecule and expression options. The atom and bond expression options passed as the second and third arguments to the method are defined in the *OEExprOpts* namespace. The expression options are used to convert the atom and bond data into expression trees which are used during the common subgraph search.

operator bool

```
operator bool() const
```

Test for successful initialization of an *OECliqueSearch* object. If initialization was attempted with an invalid SMARTS pattern or query molecule (*OEQMolBase*) then this method will return `false`. If initialization was completed successfully this method will return `true`. An *OECliqueSearch* object is considered to be uninitialized when constructed with the default constructor.

AddConstraint

```
bool AddConstraint(const OEMatchPair<OEAtomBase> &)
bool AddConstraint(const OEMatchPair<OEBondBase> &)
```

The search space of a common subgraph determination can be restricted by constraining pairs of nodes or edges (atoms or bonds) to be mapped onto one another in all subgraph solutions. Failure to satisfy atom or bond pairwise constraints will prevent any subgraph solutions from being identified. Constraints are considered satisfied in subgraphs which do not contain any constrained atoms or bonds in either the pattern or target molecules.

Both *OECliqueSearch.AddConstraint* methods return `true` if a constraint is added successfully. If the pattern atom or bond in the *OEMatchPair* does not exist as part of the query molecule created in the initialization of the *OECliqueSearch* object then *OECliqueSearch.AddConstraint* will return `false`. Multiple calls to *OECliqueSearch.AddConstraint* using the same pattern atom or bond will cause previously stored constraints to be overwritten as constraints are mutually exclusive. It is impossible to satisfy multiple simultaneous constraints for a single pattern atom or bond, hence the exclusivity.

ClearConstraints

```
void ClearConstraints()
```

Clears all prior match constraints set with the *OECliqueSearch.AddConstraint* methods.

GetMaxMatches

```
unsigned int GetMaxMatches() const
```

Returns the maximum number of subgraphs that the *OECliqueSearch* will identify before terminating the search. A value of zero indicates that no arbitrary limit has been set on the total number of subgraphs to be identified. By default, the maximum number of matches is set to `1024` This value can be set using the *OECliqueSearch.SetMaxMatches* method.

GetMinAtoms

```
unsigned int GetMinAtoms() const
```

Returns the minimum number of atoms of a subgraph that can be returned as a common subgraph. This value can be set using the *OECliqueSearch.SetMinAtoms* method.

GetPattern

```
const OEQMolBase& GetPattern() const
```

Returns a read-only reference to the query molecule (*OEQMolBase*) of *OECliqueSearch*. Const *OEQMolBase* methods can be used on the returned *OEQMolBase* reference. If the *OECliqueSearch* object has not been initialized, a reference to an empty molecule will be returned.

GetSaveRange

```
unsigned int GetSaveRange() const
```

This value can be set using the *OECliqueSearch.SetSaveRange* method. Any subgraphs found by the MCS function used by the search are discarded that have a smaller scoring value than the maximum score minus this 'range'. See *OEMCSSearch* for more details.

Init

```
bool Init(const char* smarts)
```

(Re)initializes an *OECliqueSearch* object using a 'smarts' pattern. The SMARTS pattern is parsed to create a query molecule available for subgraph matching. The method will return `true` if initialization completes successfully, and `false` upon failure.

> **Note**
>
> Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

```
bool Init(const OEQMolBase& qmol, bool perception = true)
```

(Re)initializes an *OECliqueSearch* object using a query molecule (*OEQMolBase*) reference. The method will return `true` if initialization completes successfully, and `false` upon failure.

qmol
: The query molecule of the MCS search.

perception
: If `true`, then ring perception is performed on the query molecule.

> **Note**
>
> Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

```
bool Init(const OEMolBase& mol, unsigned int atomexpr,
          unsigned int bondexpr)
```

(Re)initializes an *OECliqueSearch* object using a molecule (*OEMolBase*) and expression options used to convert the molecule into a query molecule (*OEQMolBase*). The atom and bond expression options passed as the second and third arguments to the method are defined in the *OEExprOpts* namespace. Expression options are used to convert the atom and bond data into expression trees which are used during the common subgraph search. The method will return `true` if initialization completes successfully, and `false` upon failure.

Note

Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

Match

```
OESystem::OEIterBase<OEMatchBase>* Match(const OEMolBase &)
```

Performs search in order to identify common subgraphs (cliques) shared between the passed *OEMolBase* and the query molecule which the *OECliqueSearch* object was initialized. It returns an iterator pointer (*OEIterBase*) over graph matches (*OEMatchBase*), and should be assigned to an *OEIter* < *OEMatchBase* > in order to prevent memory leaks.

If the 'uniquematch' argument passed to the methods is `true` then only the unique matches will be returned in the iterator over the matches. By definition, a match or subgraph is considered unique if it differs from all other subgraphs found previously by at least one atom or bond. Additionally, it is also considered unique if the query subgraph is mapped to a different part of the target (See example in the *Maximum Common Substructure Search* section)

SetMCSFunc

```
bool SetMCSFunc(const OEMCSFunc &)
```

Stores a copy of the passed *OEMCSFunc* functor in the *OECliqueSearch* object. The functor is then used to evaluate and order subgraphs found during a common subgraph search.

See also

- *OEMCSFunc*
- *OEMCSMaxAtoms*
- *OEMCSMaxBonds*
- *OEMCSMaxAtomsCompleteCycles*
- *OEMCSMaxBondsCompleteCycles*

SetMaxMatches

```
bool SetMaxMatches(unsigned int)
```

Sets the maximum number of common subgraph (clique) matches that will be returned by the *OECliqueSearch.Match* method. The default limit set upon construction of an *OECliqueSearch* object is `1024` matches. This value can be retrieved using the *OECliqueSearch.GetMaxMatches* method.

SetMinAtoms

```
bool SetMinAtoms(unsigned int)
```

Sets the minimum number of atoms required of a subgraph match to be returned a solution by a clique search. A single atom can be a perfectly valid common subgraph, however, for many applications such a small subgraph may not be considered useful. Setting the minimum number of atoms to an appropriate size prevents unproductive subgraph matches from being returned by the *OECliqueSearch.Match* method. The default value for the minimum number of atoms in a subgraph match is set to 1 upon construction of an *OECliqueSearch* object. This value can be retrieved using the *OECliqueSearch.GetMinAtoms* method.

SetSaveRange

```
bool SetSaveRange(unsigned int)
```

Sets the range of the clique search. This value can be retrieved using the *OECliqueSearch.GetSaveRange* method.

OEConfBase

```
class OEConfBase : public OEMolBase
```

OEConfBase is the generic conformer base class. It is an abstract base class which defines the interface for conformer implementations. Coordinates can be stored in *OEHalfFloat* (16-bit), `float` (32-bit), `double` (64-bit), or `long double` (>= 64-bit). The precision is determined by the constructor argument given to *OEMol*, taken from the *OEMCMolType* namespace. The default is 32-bit `float`.

The following methods are publicly inherited from *OEMolBase*:

operator=	*GetDimension*	*NumBonds*
operator bool	*GetEnergy*	*NumGroups*
Clear	*GetGroup*	*OrderAtoms*
ClearCoords	*GetGroups*	*OrderBonds*
Compress	*GetMaxAtomIdx*	*ResetPerceived*
Count	*GetMaxBondIdx*	*SetCoords*
CreateCopy	*GetTitle*	*SetDimension*
DeleteAtom	*HasPerceived*	*SetEnergy*
DeleteBond	*IsDataType*	*SetPerceived*
DeleteGroup	*IsDeleted*	*SetRxn*
GetAtom	*IsRxn*	*SetTitle*
GetAtoms	*NewAtom*	*Sweep*
GetBond	*NewBond*	*UnCompress*
GetBonds	*NewGroup*	
GetCoords	*NumAtoms*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

AddAtom

```
void AddAtom(OEAtomBase* atom)=0
void AddAtom(const OEAtomBase* rhs, OEAtomBase* atom)=0
```

OEConfBase.AddAtom is a helper method that the *OEMCMolBase* can call as an adjunct to *OEMolBase.NewAtom* on the molecule which contains the conformers.

> **Warning**
>
> This should not be called by a primary user.

AddBond

```
void AddBond(OEBondBase* bond)=0
```

OEConfBase.AddBond is a helper method that the *OEMCMolBase* can call as an adjunct to *OEMolBase.NewBond* on the molecule which contains the conformers.

> **Warning**
>
> This should not be called by a primary user.

Delete

```
void Delete()=0
```

The *OEConfBase.Delete* method is equivalent to call *OEMCMolBase.DeleteConf* from the parent molecule. It removes the current conformer from the *OEMCMolBase* object containing it. These functions do not invalidate current *OEIter* objects, and all calls to *OEMCMolBase.GetConfs* called after the call to *OEConfBase.Delete* will not contain the deleted conformer.

ClearCoords

```
void ClearCoords() = 0
```

Clears the coordinates, clears perception flags and sets the coordinate dimension to `0`

See also

- *OEConfBase.GetCoords*
- *OEConfBase.SetCoords*

GetCoords

```
bool GetCoords(OEPlatform::OEHalfFloat *) const = 0
bool GetCoords(float *) const = 0
bool GetCoords(double *) const = 0
bool GetCoords(long double *) const = 0
```

Fills the array passed into the function with a copy of the coordinates of the conformer. The array passed in should be at least of length *OEConfBase.GetMaxAtomIdx* * 3. The coordinates of each atom in the conformer will begin at `coords[atom->GetIdx()*3]` in the array. Changes made to the array after this function call will have no effect on the conformer.

```
bool GetCoords(const OEAtomBase *, OEPlatform::OEHalfFloat* coords) const = 0
bool GetCoords(const OEAtomBase *, float* coords) const = 0
bool GetCoords(const OEAtomBase *, double* coords) const = 0
bool GetCoords(const OEAtomBase *, long double* coords) const = 0
```

Fills the `coords` array with the coordinates of the *OEAtomBase* object. The `coords` array should be at least of length 3. Changes made to the `coord` array after this function call will have no effect on the conformer.

See also

- *OEConfBase.SetCoords*
- *OEConfBase.ClearCoords*

GetIdx

```
unsigned int GetIdx() const = 0
```

Returns the index of the conformer. This value is assigned by OEChem when a conformer is created. The conformer index is unique for the conformation of a given molecule, and is stable (not reused) for the lifetime of that *OEMCMolBase*. These values may not be assigned sequentially, and may contain gaps. Conformer indices are guaranteed to be less than *OEMCMolBase.GetMaxConfIdx*. Conformer indices are typically only used for efficiently storing data in arrays externally to the *OEMCMolBase*. The arrays can be conveniently indexed via *OEConfBase.GetIdx*. For iterating across the conformers of a molecule, use *OEMCMolBase.GetConfs* and for keeping track of a particular conformer use the conformer pointer *OEConfBase**.

GetMCMol

```
OEMCMolBase& GetMCMol() = 0
const OEMCMolBase& GetMCMol() const = 0
```

Returns the associated multi-conformer molecule that is serving as the container for the *OEConfBase*.

GetTitle

```
const char* GetTitle() const = 0
```

OEConfBase objects can have their own title. However, if they do not, they will report the title of their parent *OEMCMolBase* when the *OEConfBase.GetTitle* method is called.

See also

- *OEConfBase.HasTitle*

GetTorsion

```
double GetTorsion(OETorsion &) const = 0
double GetTorsion(const OEAtomBase* a, const OEAtomBase* b,
                  const OEAtomBase* c, const OEAtomBase* d) const = 0
```

Returns the torsion value in the conformer. The torsion can be defined by either an *OETorsion* object, or by four *OEAtomBase* objects.

See also

- *OEConfBase.SetTorsion*

GetTransform

```
const OETrans& GetTransform() const = 0
```

Returns an *OETrans*, which is a container of all the transformations which have been applied to the *OEConfBase*. This function is particularly useful when the internal representation of coordinates may not be Cartesian.

HasTitle

```
bool HasTitle() const = 0
```

Returns whether this conformer has a title set.

See also

- *OEConfBase.GetTitle*

IsDeleted

```
bool IsDeleted() const = 0
```

OEConfBase.IsDeleted is a helper function which assists other classes in maintaining functionality after an *OEConfBase* has been deleted.

See also

- *OEMolBase.IsDeleted* methods inherited from the *OEMolBase* class. That method has the purpose of identifying deleted atoms and bonds.

RemoveAtom

```
bool RemoveAtom(OEAtomBase *atom)=0
```

OEConfBase.RemoveAtom is a helper function which the *OEMCMolBase* can call as an adjunct to *OEMolBase.DeleteAtom* on the molecule which contains the conformers.

Warning

This should not be called by a primary user.

RemoveBond

```
bool RemoveBond(OEBondBase *bond)=0
```

OEConfBase.RemoveBond is a helper function which the *OEMCMolBase* can call as an adjunct to *OEMolBase.DeleteBond* on the molecule which contains the conformers.

Warning

This should not be called by a primary user.

SetCoords

```
bool SetCoords(const OEPlatform::OEHalfFloat* coords)=0
bool SetCoords(const float* coords)=0
bool SetCoords(const double* coords)=0
bool SetCoords(const long double* coords)=0
```

Sets the coordinates of a conformer. The `coords` parameter is expected to be an array of size *OEConfBase.GetMaxAtomIdx* * 3. This array should contain 3 coordinates for each atom, and they should be located in the array passed in at `coords[atom->GetIdx() * 3 + i]`, where i ranges from 0 to 3. An internal copy of the coordinates is stored in the *OEConfBase* that is independent of the `coords` array.

```
bool SetCoords(const OEAtomBase *,
               const OEPlatform::OEHalfFloat* coords)=0
bool SetCoords(const OEAtomBase *, const float* coords)=0
bool SetCoords(const OEAtomBase *, const double* coords)=0
bool SetCoords(const OEAtomBase *, const long double* coords)=0
```

Sets to coordinates of a single atom in the *OEConfBase*. The array passed in should be of length 3 and should contain the new coordinates of the atom. An internal copy of the coordinates is stored in the *OEConfBase* which is independent of the `coords` array.

See also

- *OEConfBase.GetCoords*
- *OEConfBase.ClearCoords*

SetTorsion

```
void SetTorsion(OETorsion &)=0
void SetTorsion(OEAtomBase* a, OEAtomBase* b, OEAtomBase* c,
                OEAtomBase* d, double radians)=0
```

Sets torsion value in the conformer. The torsion can be defined by either an *OETorsion* object, or by four *OEAtomBase* objects

See also

- *OEConfBase.GetTorsion*

Transform

```
void Transform(const OETrans &)=0
void Transform(const OETransBase *)=0
```

Applies the geometric transformation specified in the *OETransBase* object or the series of geometric transformations specified in the *OETrans* container to the *OEConfBase*. These functions permanently change the coordinates of the *OEConfBase*.

OEConfRMSD

```
class OEConfRMSD
```

The *OEConfRMSD* class provides fast RMSD calculation of conformers of the same molecule. During the RMSD calculation the automorphisms of the molecule graph is taken into consideration. Automorphisms are the symmetry related transformations of a molecule which can result in anomalously high RMSDs if not properly treated. For instance, t-butyl-benzene has a three-fold automorphism around the t-butyl group and a two-fold automorphism around the benzene ring.

Constructors

```
OEConfRMSD(const OEMolBase& mol,
           const OEConfRMSDOptions& opts = OEConfRMSDOptions())
```

Constructs an *OEConfRMSD* object using a *OEMolBase* and an optional *OEConfRMSDOptions*.

See also

- *OEConfRMSDOptions* class

CalculateRMSD

```
double CalculateRMSD(const OEConfBase* refconf,
                     const OEConfBase* fitconf) const
double CalculateRMSD(const double* refcoords,
                     const double* fitcoords) const
```

Calculates the RMSD between the two conformations. The inputs can be either the *OEConfRMSD* objects or their coordinates.

For the second overload, the length of the arrays should be `3*MaxAtomIdx` and should contain the Cartesian coordinates of the two conformers being assessed.

GetOptions

```
OEConfRMSDOptions GetOptions() const
```

Returns the options used to set up the *OEConfRMSD* object.

See also

- *OEConfRMSDOptions* class

IsValid

```
bool IsValid() const
```

Returns whether the *OEConfRMSD* object is valid.

MinimizeRMSD

```
double MinimizeRMSD(const OEConfBase* refconf,
                    const OEConfBase* fitconf, OETrans& trans) const
double MinimizeRMSD(const double* refcoords,
                    const double* fitcoords, OETrans& trans) const
```

The function returns the minimum RMSD between the two conformations and reports the translation and rotation, *trans*, required to give this minimum RMSD.

For the second overload, the length of the arrays should be `3*MaxAtomIdx` and should contain the Cartesian coordinates of the two conformers being assessed.

NumAutomorphs

```
unsigned int NumAutomorphs() const
```

The function returns the number of automorphisms taken into account during the RMSD calculation.

Example:

The following example finds the two conformers of a molecule that have the smallest RMSD.

```
OEMol mol = new OEMol();
OEChem.OEReadMolecule(ifs, mol);
OEConfRMSD crmsd = new OEConfRMSD(mol);

List<OEConfBase> conflist = new List<OEConfBase>();
foreach (OEConfBase conf in mol.GetConfs())
  conflist.Add(conf);

int length = conflist.Count;
double minRMSD = double.MaxValue;
uint confiIdx = 0;
uint confjIdx = 0;
for (int i = 0; i < length - 1; i++)
{
  for(int j = i + 1; j < length; j++)
  {
    OETrans trans = new OETrans();
    double dist = crmsd.MinimizeRMSD(conflist[i], conflist[j], trans);
    if (dist < minRMSD)
    {
      minRMSD = dist;
      confiIdx = conflist[i].GetIdx();
      confjIdx = conflist[j].GetIdx();
    }
  }
}

System.Console.WriteLine("The closest conformers are " + confiIdx + " and " + confjIdx +
      " with RMSD = " + minRMSD);
```

See also

- *OERMSD* function
- *OEWeightedRMSD* function

OEConfRMSDOptions

```
class OEConfRMSDOptions : public OESystem::OEOptions
```

This class represents options for setting up *OEConfRMSD* calculations.

See also

- *OEConfRMSD* class

Constructors

```
OEConfRMSDOptions()
OEConfRMSDOptions(const OEConfRMSDOptions &)
```

Default and copy constructors.

GetIgnoreHydrogens

```
bool GetIgnoreHydrogens() const
```

If `true`, hydrogen atoms are ignored when assessing automorphisms.

GetIgnoreTerminalHeavy

```
bool GetIgnoreTerminalHeavy() const
```

If `true`, terminal heavy atoms are ignored when assessing automorphisms.

GetMaxMatches

```
unsigned int GetMaxMatches() const
```

Returns the maximum number of automorphism solutions that the *OESubSearch* object will identify before terminating the search.

SetIgnoreHydrogens

```
bool SetIgnoreHydrogens(bool ignore)
```

Sets whether to ignore or consider hydrogen atoms when assessing automorphisms in the molecule graph. The default is `true`. Hydrogen atom can not be considered while ignoring terminal heavy atoms.

SetIgnoreTerminalHeavy

```
bool SetIgnoreTerminalHeavy(bool ignore)
```

Sets whether to ignore or consider terminal heavy atoms assessing automorphisms in the molecule graphs. The default is `false`. Terminal heavy atoms can not be ignored while considering hydrogens.

SetMaxMatches

```
bool SetMaxMatches(unsigned int maxmatches)
```

Sets the maximum number of subgraphs to be determined by the *OESubSearch.Match* method. Once the maximum number of subgraphs have been found, the search for additional subgraphs is terminated. By default, the maximum number of matches is set to `16384`. The method returns `false` if the maximum number of matches is zero.

OEConfTestBase

```
class OEConfTestBase
```

OEConfTestBase is the abstract base class which defines the interface for classes used to determine whether consecutive molecules read from single-conformer file-formats are combined into multi-conformer molecules.

The following classes derive from this class:

- *OEAbsCanonicalConfTest*
- *OEAbsoluteConfTest*
- *OEDefaultConfTest*
- *OEIsomericConfTest*
- *OEOmegaConfTest*

CombineMols

```
bool CombineMols(OEMCMolBase& m1, OEMolBase& m2)=0
```

Adds `m2` as a new conformer in the multi-conformer `m1` molecule. Returns a boolean indicating whether this combination was successful. This method should only be called if a previous call to *OEConfTestBase.CompareMols* previously returned `true`.

CompareMols

```
bool CompareMols(OEMolBase& m1, OEMolBase& m2) const = 0
```

This is the pure virtual function used to determine if two molecules are conformers of the same multi-conformer molecule. Users inheriting from this class must define this function. Note the molecule which is the second argument is not const. This allows the second molecule to be modified as may be necessary for it to be part of a multi-conformer molecule. It can be assumed that every molecule from an input stream will be used as argument 'm2' in this function call at least once.

CreateCopy

```
OEConfTestBase* CreateCopy() const = 0
```

Return a pointer to a copy of the derived object. This must be implemented by all derived classes.

HasCompareMols

```
bool HasCompareMols() const
```

Return whether the *OEConfTestBase.CompareMols* method has a "real" implementation and does not always return `false`. This allows *OEReadMolecule* to significantly optimize the performance of reading single conformer file formats into *OEMCMolBase* objects by avoiding copy operations. By default, this base-class method always returns `true` to be backwards compatible with sub-classes already written before this method was added. *OEDefaultConfTest.HasCompareMols* is the only **OEChem TK** implementation of this method that returns `false`, allowing significant performance optimization for the reading into *OEMol* by default.

OEConstCoords

> **Warning**
>
> This class is only usable from the OpenEye C++ toolkits.

```
template<class T>
class OEConstCoords : public detail::OECoordsBase<const T, const OEConfBase>
```

This class is used to provide convenient direct access to the memory storing coordinates for a *OEConfBase* through the *OEConfBase.GetCoordsPtr* methods. If the data type of `T` does not match the implementation of *OEConfBase*, this class makes a copy of the coordinates with *OEConfBase.GetCoords* instead.

> **See also**
>
> This class only provides `const` read-only access, use *OEMutableCoords* for write-access as well.

Constructors

```
OEConstCoords(const OEConfBase* conf)
```

Construct a read-only view of the coordinates in the conformer `conf`.

GetPtr

```
const T* GetPtr() const
```

Returns a const pointer of type `T` to the coordinates of the conformer used in the constructor.

OECoordArray

```
class OECoordArray
```

This class represents *OECoordArray*.

Constructors

```
OECoordArray()
OECoordArray(const OECoordArray& rhs)
OECoordArray(const OEChem::OEMolBase& mol)
```

Default and copy constructors.

OECreateSubSearchDatabaseOptions

Attention

This API is currently available in C++ and Python.

```
class OECreateSubSearchDatabaseOptions
```

This class represents the *OECreateSubSearchDatabaseOptions* class that encapsulates properties that determine how a substructure search database is generated when calling the *OECreateSubSearchDatabaseFile* function.

The *OECreateSubSearchDatabaseOptions* class stores the following properties:

Property	Default	Get method	Set method	Corresponding namespace / class / type
screen type		*GetScreenType*	*Constructor*	*OESubSearchScreenTypeBase* class
database sorting	`true`	*GetSortByBitCou*	*SetSortByBitCou*	boolean
handling molecule titles	`false`	*GetKeepTitle*	*SetKeepTitle*	boolean
number of processors	`0` (0 means *OEGetNumProcessors*)	*GetNumProcessor*	*SetNumProcessor*	unsigned integer

See also

- *OECreateSubSearchDatabaseFile* function
- *OESubSearchDatabase* class

Code Example

- *Generate database for fast substructure search* example

Constructors

```
OECreateSubSearchDatabaseOptions(const OESubSearchScreenTypeBase *)
```

Constructs an *OESubSearchScreenTypeBase* object with the given screen type.

GetKeepTitle

```
bool GetKeepTitle() const
```

Returns whether the molecule titles are kept in the output substructure search database. The default is `false`.

See also

- *OECreateSubSearchDatabaseOptions.SetKeepTitle* method

GetNumProcessors

```
unsigned GetNumProcessors() const
```

Returned the number of processors used during the database generation.

See also

- *OECreateSubSearchDatabaseOptions.SetNumProcessors* method

GetScreenType

```
const OESubSearchScreenTypeBase* GetScreenType() const
```

Returns the type of the screen that will be generated for each molecule in substructure search database.

See also

- *OESubSearchScreenTypeBase* class

GetSortByBitCounts

```
bool GetSortByBitCounts() const
```

Returns whether to sort the molecules based on their screen bit counts. The default is `true`.

See also

- *OECreateSubSearchDatabaseOptions.SetSortByBitCounts* method

SetKeepTitle

```
void SetKeepTitle(const bool keep)
```

Sets whether to keep molecule titles taken from the input file as unique identifiers. If `false`, a 16 character long UUID will be assigned for each molecule as a new title.

Hint

If the original molecule titles are kept, these titles should be unique. *OECreateSubSearchDatabaseFile* will not check or throw any warnings for non-unique title when generating the substructure database files.

By keeping the titles when the database is generated, an external id (such as corporate or vendor id) can be associate with each molecule in the substructure search database. When searching the database, the titles of the matched molecules can be retrieved by the *OESubSearchDatabase.GetMatchTitles* method.

SetNumProcessors

```
void SetNumProcessors(const unsigned)
```

Sets the number of threads that are used to generate the substructure search database when calling the *OECreateSubSearchDatabaseFile* function. When set to `0`, the number of processors used will be the number returned by the *OEGetNumProcessors* function.

SetSortByBitCounts

```
void SetSortByBitCounts(const bool sort)
```

Sets whether to sort the molecules based on their screen bit counts.

Hint

The *OESubSearchDatabase* works with both sorted and unsorted input database files. While sorting the database at generation can take slightly more time, this pays off at search time. Based on the query molecule, significant portion of the screen matches can be skipped without losing any matches when using a sorted input database.

OECSVHeader

```
class OECSVHeader
```

This class stores the column names to be read or written to a *OEFormat.CSV* file. It is typically retrieved from an *oemolstreambase* through the *oemolstreambase.GetCSVHeader* method. The object available from that method is initialized during the first read or write operation on that stream.

See also

OECreateCSVHeader for creating a *OECSVHeader* from the SD data tag in a molecule.

Constructors

```
OECSVHeader()
OECSVHeader(const OECSVHeader& rhs)
OECSVHeader(const std::vector<std::string>& columnNames)
```

The default constructors initializes the object to an empty state. The copy constructor can be used to make a copy from about *OECSVHeader* object. The class can also be constructed directly from a `std::vector` of `std::string`.

operator bool

```
operator bool() const
```

Returns `false` if this class has not been initialized with any column names yet. Returns `true` if this class does contain column names.

Clear

```
void Clear()
```

Resets the class to an empty uninitialized state.

GetColumnName

```
const std::string& GetColumnName(unsigned int fieldIdx) const
```

Return the column name for the particular index, starting at zero. If `fieldIdx` is not less than *OECSVHeader.GetNumColumns*, a reference to an empty string is returned.

GetColumnNames

```
OESystem::OEIterBase<const std::string>* GetColumnNames() const
```

Returns an iterator over all the column names stored in this class.

GetNumColumns

```
unsigned int GetNumColumns() const
```

Returns the number of column names stored in this class.

IsValid

```
bool IsValid() const
```

Returns `false` if this class has not been initialized with any column names yet. Returns `true` if this class does contain column names.

OEDefaultConfTest

```
class OEDefaultConfTest : public OEConfTestBase
```

This class represents *OEDefaultConfTest*.

This is the default implementation of *OEConfTestBase*. It never combines connection tables into multi-conformer molecules. This function is set as the conformer test to use by all of the *oemolistream* constructors.

The following methods are publicly inherited from *OEConfTestBase*:

CombineMols	*CompareMols*	*CreateCopy*

CombineMols

```
bool CombineMols(OEMCMolBase& m1, OEMolBase& m2)
```

Adds `m2` as a new conformer in `m1` and returns whether the combination was successful.

CompareMols

```
bool CompareMols(OEMolBase& m1, OEMolBase& m2) const
```

This method always returns `false`.

CreateCopy

```
OEConfTestBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEDefaultConfTest* object is dynamically allocated and owned by the caller.

HasCompareMols

```
bool HasCompareMols() const
```

Always returns `false` for this object to indicate that the implementation of the *OEConfTestBase.CompareMols* method is "trivial" and always returns `false`, indicating molecule copying can be elided in *OEReadMolecule*.

OEEuler

```
class OEEuler : public OETransBase
```

This class represents *OEEuler*.

The following methods are publicly inherited from *OETransBase*:

CreateCopy	*GetInverse*
Extract	*Transform*

Constructors

```
OEEuler()
OEEuler(const float* angles)
OEEuler(const double* angles)
OEEuler(float phi, float theta, float psi)
OEEuler(double phi, double theta, double psi)
```

Default and copy constructors.

CreateCopy

```
OETransBase* CreateCopy() const
```

Extract

```
void Extract(float* r, float* t) const
void Extract(float* r, double* t) const
void Extract(double* r, float* t) const
void Extract(double* r, double* t) const
```

GetAngles

```
void GetAngles(float* angles) const
void GetAngles(double* angles) const
```

GetInverse

```
OETransBase* GetInverse() const
```

GetPhi

```
double GetPhi() const
```

GetPsi

```
double GetPsi() const
```

GetTheta

```
double GetTheta() const
```

SetPhi

```
void SetPhi(double radians)
```

SetPsi

```
void SetPsi(double radians)
```

SetTheta

```
void SetTheta(double radians)
```

OEExprBase

```
class OEExprBase
```

This class represents *OEExprBase*.

The *OEExprBase* is an abstract class which defines an interface through which boolean expressions can test equivalence of nodes (atoms) and edges (bonds) during graph match processes. The expression class interface simplifies what is normally a binary tree representation of attributes which atoms and bonds must (not) have in order to be called equivalent to a corresponding node or edge. Derived classes may be used to extend the graph matching algorithms in *OEChem* to compare attributes beyond the pre-defined expressions in *OEChem*. In addition, user-defined expression classes may take full advantage of the arbitrary data storage mechanism which is part of the *OEBase* class. Expression equivalence could then be performed on dynamic user-defined data extensions to atoms and bonds.

Constructors

```
OEExprBase()
```

Default and copy constructors.

CreateCopy

```
OEExprBase* CreateCopy() const = 0
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEExprBase* object is dynamically allocated and owned by the caller.

GetType

```
unsigned int GetType() const = 0
```

This pure virtual method is designed to return a unique identifier for each *OEExprBase* implementation. A listing of type identifiers returned by classes in *OEChem* which are derived from the *OEExprBase* class can be found in the *OEExprType* namespace. User-defined implementations of the *OEExprBase* class should return value for this method which does not collide with types returned by implementations provided by *OEChem*.

IsEquivalent

```
const OEFuzzy& IsEquivalent(const OEBondBase *) const = 0
const OEFuzzy& IsEquivalent(const OEAtomBase *) const = 0
```

These pure virtual methods provide a common interface for testing attributes of atoms (*OEAtomBase*) and bonds (*OEBondBase*). Any method which does not modify an atom or bond may be called in an implementation of an *OEExprBase.IsEquivalent* method. The return value must be one of the *OEFuzzy* predefined instances of true (*OEFuzzVal.True*), false (*OEFuzzVal.False*), or maybe (*OEFuzzVal.Maybe*).

OEExtractRingTemplateOptions

```
class OEExtractRingTemplateOptions
```

This class represents *OEExtractRingTemplateOptions* that stores the parameters that are used when extracting ring templates using *OEExtractRingTemplates* function

Note

We recommend using the *OE2DRingDictionary* class for generating a ring template dictionary.

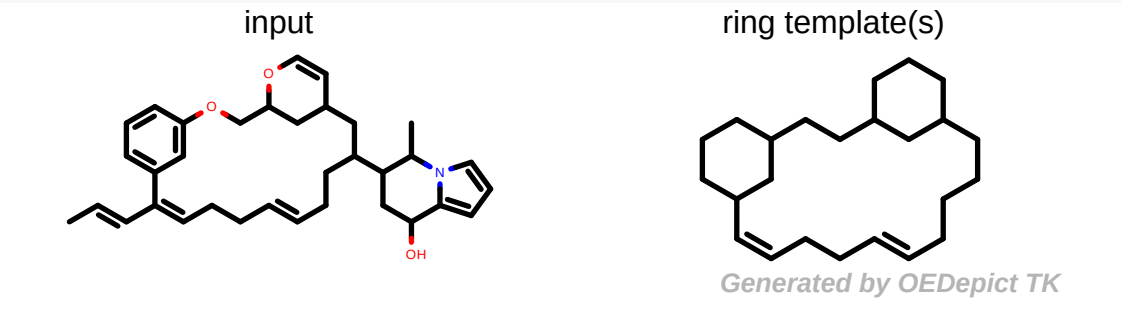


Fig. 3: **Example of extracting ring templates with default parameters**

See also

- *OEExtractRingTemplates* function

Constructors

```
OEExtractRingTemplateOptions()
```

Default constructor.

```
OEExtractRingTemplateOptions(const OEExtractRingTemplateOptions& rhs)
```

Copy constructor.

GetRetainBondStereo

```
bool GetRetainBondStereo() const
```

Returns whether the cis/trans bond stereo configurations are kept in macro-cycles when extracting ring systems.

See also

- *OEExtractRingTemplateOptions.SetRetainBondStereo* method

GetRetainExistingTemplates

```
bool GetRetainExistingTemplates() const
```

Returns whether to return or ignore ring templates that have been already exist in the 2D coordinate generation system of OEChem.

See also

- *OEExtractRingTemplateOptions.SetRetainExistingTemplates* method

SetRetainBondStereo

```
void SetRetainBondStereo(bool retain)
```

Sets whether the cis/trans bond stereo configurations are kept in macro-cycles when extracting ring systems. By default, the bond stereo configurations in macro-cycles are retained.

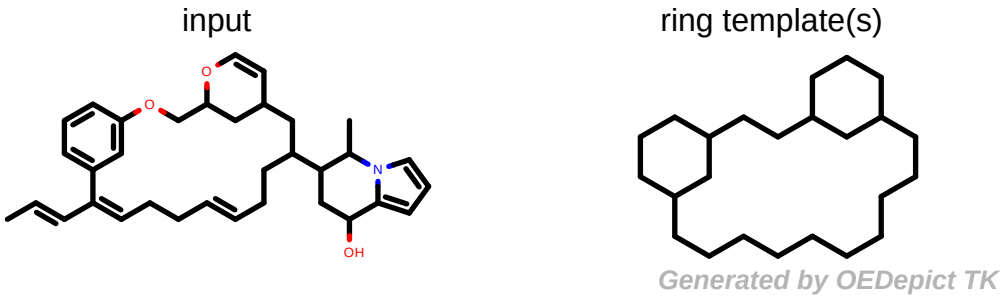


Fig. 4: **Example of extracting ring templates with removing bond stereo configurations**

See also

- *OEExtractRingTemplateOptions.GetRetainBondStereo* method

SetRetainExistingTemplates

```
void SetRetainExistingTemplates(bool retain)
```

Sets whether to return or ignore ring templates that have been already exist in the 2D coordinate generation system of OEChem. By default, the existing ring templates are ignored.

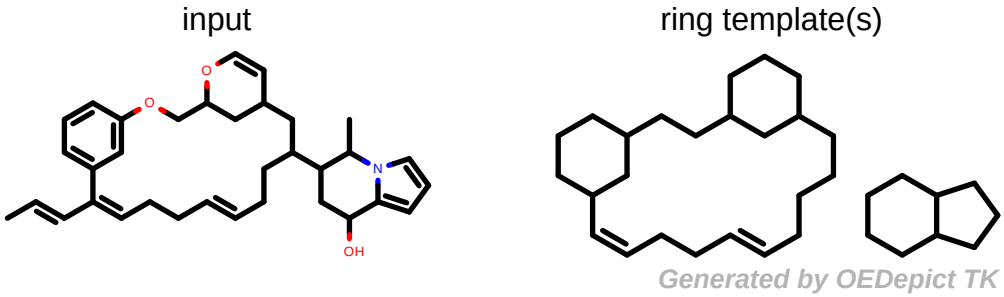


Fig. 5: **Example of extracting ring templates with keeping existing ring templates**

See also

- *OEExtractRingTemplateOptions.GetRetainExistingTemplates* method

OEFileStringParameter

```
class OEFileStringParameter : public OESystem::OEStringParameter
```

The *OEFileStringParameter* represents parameter that has `string` value representing a readable file format for modeling objects, such as a molecule or a design unit.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEFileStringParameter()
OEFileStringParameter(const std::string& name,
                      const unsigned fileType,
                      const unsigned fileTypeFlavor = 0u)
OEFileStringParameter(const std::string& name,
                      const std::string& fileExt)
OEFileStringParameter(const OEFileStringParameter&)
```

Default and copy constructors.

Constructs an *OEFileStringParameter* instance using the specified set of parameters. The `fileType` argument refers to constants defined in the *OEFileStringType* namespace. The `fileTypeFlavor` argument refers to constants defined in the *OEFileStringTypeFlavor* namespace.

OEFuzzy

```
class OEFuzzy
```

This class represents *OEFuzzy*.

The *OEFuzzy* class represents fuzzy logic used in expression tests.

Constructors

```
OEFuzzy()
```

Default constructor.

```
OEFuzzy(const unsigned int v)
```

Construct an *OEFuzzy* object given with an integer value. The integer values should be one of the values give in the *OEFuzzVal* namespace. Three pre-defined global constant fuzzy values (*OEFuzzVal.True*, *OEFuzzVal.False*, and *OEFuzzVal.Maybe*) are define in *OEChem* which represent the fuzzy values of true, false, and maybe, respectively.

Default and copy constructors.

operator bool

```
operator bool() const
```

Boolean test operator. Both fuzzy true (*OEFuzzVal.True*) and maybe (*OEFuzzVal.Maybe*) values will return `true`. A fuzzy false(*OEFuzzVal.False*) value will return `false`.

OEGetNbrAtom

```
class OEGetNbrAtom : public OESystem::OEUnaryFunction<OEChem::OEBondBase*,␣
↪OEChem::OEAtomBase*>
```

This class represents *OEGetNbrAtom* functor that, when constructed with one atom and passed a bond, will return the neighbor atom which is across the bond from the construction atom.

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	CreateCopy

Constructors

```
OEGetNbrAtom(const OEChem::OEAtomBase* atom)
```

Constructs the functor with atom (*OEAtomBase*) as the reference atom used to find neighbors.

operator()

```
result_type operator()(const argument_type &) const
```

Returns the *OEAtomBase* pointer that returned by the *OEBondBase.GetNbr* (atom), where 'atom' is the construction *OEAtomBase* argument.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase *, OEChem::OEAtomBase *>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEGetNbrAtom* object is dynamically allocated and owned by the caller.

OEGraphMol

```
class OEGraphMol
```

This class represents *OEGraphMol* that is a single-conformer concrete implementation of the *OEMolBase* base class. It is recommended to use in cheminformatics applications.

See also

Molecules chapter

The following methods are publicly inherited from *OEMolBase*:

operator=	GetDimension	NumBonds
operator bool	GetEnergy	NumGroups
Clear	GetGroup	OrderAtoms
ClearCoords	GetGroups	OrderBonds
Compress	GetMaxAtomIdx	ResetPerceived
Count	GetMaxBondIdx	SetCoords
CreateCopy	GetTitle	SetDimension
DeleteAtom	HasPerceived	SetEnergy
DeleteBond	IsDataType	SetPerceived
DeleteGroup	IsDeleted	SetRxn
GetAtom	IsRxn	SetTitle
GetAtoms	NewAtom	Sweep
GetBond	NewBond	UnCompress
GetBonds	NewGroup	
GetCoords	NumAtoms	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEGraphMol(const OEMolBase &)
OEGraphMol(const OEGraphMol &)
OEGraphMol(const OEMolBase &, unsigned int type)
OEGraphMol(const OEGraphMol &, unsigned int type)
OEGraphMol(unsigned int type = OEMolBaseType::OEDefault)
OEGraphMol(const OEMol &,
           unsigned int type = OEMolBaseType::OEDefault)
OEGraphMol(const OEQMol &,
           unsigned int type = OEMolBaseType::OEDefault)
OEGraphMol(const OEQMolBase &,
           unsigned int type = OEMolBaseType::OEDefault)
OEGraphMol(const OEMCMolBase &,
           unsigned int type = OEMolBaseType::OEDefault)
```

Default and copy constructors.

SCMol

```
const OEMolBase& SCMol() const
OEMolBase& SCMol()
```

OEGroupBase

```
class OEGroupBase  : public OESystem::OEBase
```

The *OEGroupBase* class is the abstract interface for representing a 'group' container that can store atom and bond pointers of a molecule.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

AddAtom

```
bool AddAtom(OEAtomBase* atom)=0
```

Adds an atom to the group. A group can hold only atoms that belong to the same molecule on which the group is created.

See also

- *OEGroupBase.GetParent* method

AddBond

```
bool AddBond(OEBondBase* bond)=0
```

Adds a bond to the group. A group can hold only bonds that belong to the same molecule on which the group is created.

See also

- *OEGroupBase.GetParent* method

Clear

```
void Clear()=0
```

This method removes all atoms (*OEAtomBase*) and bonds (*OEBondBase*) that are stored in the group.

DeleteAtom

```
bool DeleteAtom(OEAtomBase* atom)=0
```

Deletes an atom from a group. Returns `true` if the atom was successfully removed.

DeleteBond

```
bool DeleteBond(OEBondBase* bond)=0
```

Deletes a bond from a group. Returns `true` if the bond was successfully removed.

GetAtoms

```
OESystem::OEIterBase<OEAtomBase>* GetAtoms() const = 0
```

Returns an iterator over all the atoms in the group.

```
OESystem::OEIterBase<OEAtomBase>* GetAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase> &) const = 0
```

Returns an iterator over all of the atoms in the group that match the specified atom predicate (*OEUnaryPredicate*).

GetBonds

```
OESystem::OEIterBase<OEBondBase>* GetBonds() const = 0
```

Returns an iterator over all the bonds in the group.

```
OESystem::OEIterBase<OEBondBase>* GetBonds(const OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase> &) const = 0
```

Returns an iterator over all bonds in the group that match the specified bond predicate (*OEUnaryPredicate*).

GetGroupDefinition

```
bool GetGroupDefinition(OEChem::OEAtomBondSet &) const = 0
```

Returns the entire group definition as an *OEAtomBondSet*.

GetParent

```
const OEMolBase* GetParent() const = 0
```

Returns the parent molecule of a group. All *OEGroupBases* are created as a container of an *OEMolBase*. This method can be used to determine the molecule of which atoms and bonds the group stores. The 'parent molecule' property of a group cannot be altered, as its determined at the point the group is created.

GetGroupType

```
unsigned int GetGroupType() const = 0
```

Returns the type of the group.

See also

- *OEGroupBase.SetGroupType* methods
- *OEGroupType* namespace for the built-in group types

HasAtom

```
bool HasAtom(const OEAtomBase* atom) const = 0
```

Returns `true` if the given atom belongs to the group.

HasBond

```
bool HasBond(const OEBondBase* bond) const = 0
```

Returns `true` if the given bond belongs to the group.

IsDeleted

```
bool IsDeleted() const = 0
```

Returns `true` if the group has been deleted or if all atom and bonds have been removed from the group i.e. if the group is empty.

IsEmpty

```
bool IsEmpty() const = 0
```

Returns `true` if all atom and bonds have been removed from the group, i.e. if the group is empty.

NumAtoms

```
unsigned int NumAtoms() const = 0
```

Returns the number of atoms in the group.

NumBonds

```
unsigned int NumBonds() const = 0
```

Returns the number of bonds in the group.

SetGroupType

```
void SetGroupType(unsigned int)=0
```

Sets the type of the group.

See also

- *OEGroupBase.GetGroupType* method
- *OEGroupType* namespace for the built-in group types

Sweep

```
bool Sweep()=0
```

Removes atoms and bonds from the group for which the *OEAtomBase.IsDeleted* method and the *OEBondBase.IsDeleted* method returns `true`, respectively.

OEHasAlphaBetaUnsat

```
class OEHasAlphaBetaUnsat : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasAlphaBetaUnsat*.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	CreateCopy	CreatePredicateCopy

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	CreateCopy

Constructors

```
OEHasAlphaBetaUnsat(bool includePandS = true)
```

Default and copy constructors.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

OEHasAnisou

```
class OEHasAnisou : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents the *OEHasAnisou* functor that identifies atoms (*OEAtomBase*) that have associated anisotropic B-factor parameters.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

Method returning `true` if anisotropic B-factor parameters are associated with the input atom.

See also

- *OEAnisoUij*
- *OEGetAnisou*
- *OESetAnisou*

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasAnisou* object is dynamically allocated and owned by the caller.

OEHasAtomicNum

```
class OEHasAtomicNum : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasAtomicNum* functor that identifies atoms (*OEAtomBase*) with a specific atomic number.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasAtomicNum(unsigned int an)
```

Constructs the functor with the internal atomic number.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the atomic number returned by the *OEAtomBase.GetAtomicNum* method for the given *OEAtomBase* object is equivalent to the atomic number with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasAtomicNum* object is dynamically allocated and owned by the caller.

OEHasAtomInGroup

```
class OEHasAtomInGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents *OEHasAtomInGroup* functor that identifies groups (*OEGroupBase*) that store a specific atom (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasAtomInGroup(const OEAtomBase* atom)
```

Constructs the functor with the given atom.

operator()

```
bool operator()(const OEGroupBase& group) const
```

Returns `true`, if the atom with which the functor is constructed is stored in the group.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasAtomInGroup* object is dynamically allocated and owned by the caller.

OEHasAtomName

```
class OEHasAtomName : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasAtomName* functor that identifies atoms (*OEAtomBase*) with a specific name.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasAtomName(const char* name, const bool trimspaces = false)
```

Constructs the functor with the provided name. The trimspaces allows matching atoms with names by first trimming off white spaces.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the atom name returned by the *OEAtomBase.GetName* method for the given *OEAtomBase* object is equivalent to the name with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasAtomName* object is dynamically allocated and owned by the caller.

OEHasAtomStereoSpecified

```
class OEHasAtomStereoSpecified : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasAtomStereoSpecified* functor that identifies atoms (*OEAtomBase*) which have stereo specified.

See also

- stereo atom

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasAtomStereoSpecified()
```

Default constructor. Constructs the functor so that atoms (*OEAtomBase*) with any stereo type will be identified (*OEAtomStereo.All*).

```
OEHasAtomStereoSpecified(unsigned int v)
```

Constructs the functor so that atoms (*OEAtomBase*) with the given stereo type will be identified.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.HasStereoSpecified* method returns `true` for the given *OEAtomBase* object for the stereo type with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasAtomStereoSpecified* object is dynamically allocated and owned by the caller.

OEHasBondInGroup

```
class OEHasBondInGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents *OEHasBondInGroup* functor that identifies groups (*OEGroupBase*) that store a specific bond (*OEBondBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasBondInGroup(const OEBondBase* bond)
```

Constructs the functor with the given bond.

operator()

```
bool operator()(const OEGroupBase& group) const
```

Returns `true`, if the bond with which the functor is constructed is stored in the group.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasBondInGroup* object is dynamically allocated and owned by the caller.

OEHasBondStereoSpecified

```
class OEHasBondStereoSpecified : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEHasBondStereoSpecified* functor that identifies bonds (*OEBondBase*) which have stereo specified.

See also

- stereo bond

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	CreateCopy

Constructors

```
OEHasBondStereoSpecified(unsigned int v = OEBondStereo::All)
```

Constructs the functor so that bonds (*OEBondBase*) with the given stereo type will be identified.

operator()

```
bool operator()(const OEBondBase& bond) const
```

Returns `true`, if the *OEBondBase.HasStereoSpecified* method returns `true` for the given *OEBondBase* object for the stereo type with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasBondStereoSpecified* object is dynamically allocated and owned by the caller.

OEHasChainID

```
class OEHasChainID : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasChainID* functor that identifies atoms (*OEAtomBase*) with a specific chain ID.

See also

Residue Data Functors section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	CreateCopy	CreatePredicateCopy

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	CreateCopy

Constructors

```
OEHasChainID(const char id)
```

Constructs the functor with the internal chain id.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if *OEAtomGetResidue* (atom). *GetChainID()* is equivalent to the chain id with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasChainID* object is dynamically allocated and owned by the caller.

OEHasFormalCharge

```
class OEHasFormalCharge : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasFormalCharge* functor that identifies atoms (*OEAtomBase*) with a specific formal charge.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasFormalCharge(int c)
```

Constructs the functor with the given formal charge.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the charge returned by the *OEAtomBase.GetFormalCharge* method for the given *OEAtomBase* object is equivalent to the charge with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasFormalCharge* object is dynamically allocated and owned by the caller.

OEHasFragmentNumber

```
class OEHasFragmentNumber : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasFragmentNumber* functor that identifies atoms (*OEAtomBase*) with a specific fragment number.

See also

Residue Data Functors section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasFragmentNumber(unsigned int num)
```

Constructs the functor with the internal fragment number.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if *OEAtomGetResidue* (atom). *GetFragmentNumber()* is equivalent to the number with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasFragmentNumber* object is dynamically allocated and owned by the caller.

OEHasGroupType

```
class OEHasGroupType : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents *OEHasGroupType* functor that identifies groups (*OEGroupBase*) with a specific type.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator() *CreateCopy*

Constructors

```
OEHasGroupType(unsigned int type)
```

Constructs the functor with the given type.

operator()

```
bool operator()(const OEGroupBase& group) const
```

Returns `true`, if the number returned by the *OEGroupBase.GetGroupType* method is equivalent to the number with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasGroupType* object is dynamically allocated and owned by the caller.

OEHasHvyDegree

```
class OEHasHvyDegree : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasHvyDegree* functor that identifies atoms (*OEAtomBase*) with a specific number of heavy atom neighbors.

operator() *CreateCopy* *CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator() *CreateCopy*

Constructors

```
OEHasHvyDegree(unsigned int h)
```

Constructs the functor with the given number of heavy atom neighbor.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the value returned by the *OEAtomBase.GetHvyDegree* method for the given *OEAtomBase* object is equivalent to the value with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasHvyDegree* object is dynamically allocated and owned by the caller.

OEHasIdx

```
template<class Arg>
class OEHasIdx : public OESystem::OEUnaryPredicate<Arg>
```

This class represents *OEHasIdx* template functor that identifies object that provides a `GetIdx` method with a specific index.

See also

- *Indices for Molecule Lookup Considered Harmful* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following specializations exist for this template:

Specialized Functor	Definition
OEHasAtomIdx	*OEHasIdx* <*OEAtomBase*>
OEHasBondIdx	*OEHasIdx* <*OEBondBase*>
OEHasConfIdx	*OEHasIdx* <*OEConfBase*>

Constructors

```
OEHasIdx(unsigned int idx)
```

Constructs the functor with the internal index number.

operator()

```
bool operator()(const Arg& arg) const
```

Returns `true`, if the index returned by the `Arg.GetIdx()` method is equivalent to the number with which the functor is constructed.

CreateCopy

```
OETYPENAME OESystem::OEUnaryFunction<Arg, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasIdx* object is dynamically allocated and owned by the caller.

OEHasMapIdx

```
class OEHasMapIdx : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasMapIdx* functor that identifies atoms (*OEAtomBase*) with any or a specific map index.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasMapIdx(unsigned int idx = 0)
```

Constructs the functor with the internal map index. With no arguments, this constructs a functor which returns `true` for any atom with a non-zero map index. When constructed with a non-zero integer, it constructs a functor which return `true` for atoms which have the same map index as the constructor argument.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the index returned by the *OEAtomBase.GetMapIdx* method for the given *OEAtomBase* object is equivalent to the non-zero index number with which the functor is constructed.

Returns `true`, if the index returned by the *OEAtomBase.GetMapIdx* method for the given *OEAtomBase* object is non-zero when the index number with which the functor is constructed is `0`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasMapIdx* object is dynamically allocated and owned by the caller.

OEHasOrder

```
class OEHasOrder : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEHasOrder* functor that identifies bonds (*OEBondBase*) with a specific bond order.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasOrder(unsigned int o)
```

Constructs the functor with the internal bond order.

operator()

```
bool operator()(const OEBondBase& bond) const
```

Returns `true`, if the bond order returned by the *OEBondBase.GetOrder* method for the given *OEBondBase* object is equivalent to the bond order with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasOrder* object is dynamically allocated and owned by the caller.

OEHasResidueNumber

```
class OEHasResidueNumber : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasResidueNumber* functor that identifies atoms (*OEAtomBase*) with a specific residue number.

See also

Residue Data Functors section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasResidueNumber(int num)
```

Constructs the functor with the internal residue number.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if *OEAtomGetResidue* (atom). *GetResidueNumber()* is equivalent to the residue number with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasResidueNumber* object is dynamically allocated and owned by the caller.

OEHydrogenCounter

```
class OEHydrogenCounter
```

The *OEHydrogenCounter* class is a utility class used for counting the implicit and explicit hydrogens within a molecule.

Constructors

```
OEHydrogenCounter(unsigned int explicitHCount = 0,
                  unsigned int implicitHCount = 0)
```

Constructs a counter with the specified explicit and implicit counts.

GetImplicitHCount

```
unsigned int GetImplicitHCount()
```

Returns the implicit hydrogen count stored in the counter.

GetExplicitHCount

```
unsigned int GetExplicitHCount()
```

Returns the explicit hydrogen count stored in the counter.

See also

- *OEGetHydrogenCounts*

OEIdxSelected

```
template<class Arg>
class OEIdxSelected : public OESystem::OEUnaryPredicate<Arg>
```

This class represents *OEIdxSelected*, a template functor that identifies objects which provide a `GetIdx` method and have been selected by index in the boolean *array* or *vector* or *OEBitVector* used to initialize the class.

See also

- *Indices for Molecule Lookup Considered Harmful* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following specializations exist for this template:

Specialized Functor	Definition
OEAtomIdxSelected	*OEIdxSelected* <*OEAtomBase*>
OEBondIdxSelected	*OEIdxSelected* <*OEBondBase*>
OEConfIdxSelected	*OEIdxSelected* <*OEConfBase*>

Constructors

```
OEIdxSelected(const OESystem::OEBitVector& bv)
```

Construct the functor from an *OEBitVector* (call it *bv*) where the selection state of the object with internal index *i* is stored at *bv[i]* (*e.g.* `bv.SetBitOn(atom.GetIdx())`). The caller is responsible for making sure that the input *OEBitVector* is sufficient size to refer to the maximum index value of the objects it will be applied to.

The selection states are copied into an internal representation and the *OEBitVector* can be deleted before the functor is called.

operator()

```
bool operator()(const Arg& a) const
```

Returns `true`, if the state of the *OEBitVector* used to construct the functor at the index returned by the `Arg.GetIdx()` method is `true`.

CreateCopy

```
OETYPENAME OESystem::OEUnaryFunction<Arg, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIdxSelected* object is dynamically allocated and owned by the caller.

OEInChIOptions

```
class OEInChIOptions
```

This class is used to control the output for the *OEFormat.INCHI* and *OEFormat.INCHIKEY* formats. The options have a direct mapping between the flavors in the *OEOFlavor.INCHI* and *OEOFlavor.INCHIKEY*. The options also have a direct mapping into the options provided by the InChI library.

See also

- InChI API Reference documentation
- InChI Technical Manual documentation

Constructors

```
OEInChIOptions(unsigned int flavor = OEOFlavor::INCHI::Default)
```

The default constructor initializes the options class to the flavors indicated by the *OEOFlavor.INCHI.Default* constant.

```
OEInChIOptions(const OEInChIOptions &)
```

Copy constructor.

GetChiral

```
bool GetChiral() const
```

Returns a bool stating whether to use absolute stereochemistry.

GetFixedHLayer

```
bool GetFixedHLayer() const
```

Returns whether to include the fixed H-layer.

GetHydrogens

```
bool GetHydrogens() const
```

Returns whether to add H according to usual valences.

GetNonPolymerPseudoAtoms

```
bool GetNonPolymerPseudoAtoms() const
```

Returns the state of the flag that allows non-polymer **atoms* (*OEElemNo.Du*) to be present in the structure used for InChI generation. Setting this flag results in a nonstandard InChI version. By default, no **atoms* are allowed. Currently there is no support for polymer InChIs in OEChem.

GetRacemicStereo

```
bool GetRacemicStereo() const
```

Returns whether to export all stereo as racemic stereo in the InChI string. By default, all stereo is exported as absolute stereo.

GetReconnectedMetals

```
bool GetReconnectedMetals() const
```

Returns whether to include reconnected metals.

GetRelativeStereo

```
bool GetRelativeStereo() const
```

Returns whether to export all stereo as relative stereo in the InChI string. By default, all stereo is exported as absolute stereo.

GetStereo

```
bool GetStereo() const
```

Returns whether the stereo information in the input structure is ignored.

GetTimeout

```
unsigned int GetTimeout() const
```

Returns the setting for the maximum number of seconds the InChI generation is allowed to take. By default, a 60 second execution limit is used.

GetUndefinedStereo

```
bool GetUndefinedStereo() const
```

Returns the setting that controls explicit undefined stereo annotations in the generated InChI identifier. Default, false.

SetChiral

```
void SetChiral(bool val)
```

If set to true, use absolute stereochemistry, else relative stereochemistry (corresponds to InChI option "ChiralFlagOn").

SetFixedHLayer

```
void SetFixedHLayer(bool val)
```

If set to true, do not allow for mobile hydrogens e.g. in tautomers (corresponds to InChI option "FixedH").

SetHydrogens

```
void SetHydrogens(bool val)
```

If set to true, hydrogens are added according to usual valences (corresponds to the inverse of the InChI option "DoNotAddH").

SetNonPolymerPseudoAtoms

```
void SetNonPolymerPseudoAtoms(bool val)
```

If set to true, non-polymer **atoms* (*OEElemNo.Du*) are allowed to be present in the structure used for InChI generation. Setting this flag results in a nonstandard InChI version. By default, no **atoms* are allowed. Currently there is no support for polymer InChIs in OEChem.

SetRacemicStereo

```
void SetRacemicStereo(bool val)
```

If set to true, export all stereo as racemic stereo (corresponds to InChI option "SRac"). Enabling the racemic stereo option takes precedence over the option setting to disable *OEInChIOptions.SetStereo*.

SetReconnectedMetals

```
void SetReconnectedMetals(bool val)
```

If set to true, include reconnected metals (corresponds to InChI option "RecMet").

SetRelativeStereo

```
void SetRelativeStereo(bool val)
```

If set to true, export all stereo as relative stereo (corresponds to InChI option "SRel"). Enabling the relative stereo option takes precedence over other option settings of *OEInChIOptions.SetRacemicStereo* or (disabled) *OEInChIOptions.SetStereo*.

SetStereo

```
void SetStereo(bool val)
```

If set to true, stereo information is used from input structure (corresponds to the inverse of the InChI option "SNon").

SetTimeout

```
void SetTimeout(unsigned int val)
```

Sets the InChI execution time cutoff to `val` seconds. Setting a value of `0` means unlimited, and by default a `60` second limit is used.

SetUndefinedStereo

```
void SetUndefinedStereo(bool val)
```

If set to true, requests explicit undefined stereo annotations in the generated InChI and InChIKey identifiers. Enabling this flag always results in a nonstandard InChI.

OEInSamePart

```
class OEInSamePart : public OESystem::OEBinaryPredicate<OEChem::OEAtomBase,␣
↪OEChem::OEAtomBase>
```

This class represents *OEInSamePart*.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEInSamePart(const OEInSamePart& rhs)
OEInSamePart(unsigned int* parts, unsigned int size,
             bool boolval = true)
```

Default and copy constructors.

operator()

```
bool operator()(const OEAtomBase& a1, const OEAtomBase& a2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEChem::OEAtomBase , OEChem::OEAtomBase , bool>* CreateCopy()␣
↪const
```

OEIsAmideBond

```
class OEIsAmideBond : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents the *OEIsAmideBond* functor that identifies amide bonds.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsAmideBond(bool includeCappingGroups = true)
```

Default constructor.

operator()

```
bool operator()(const OEChem::OEBondBase& bond) const
```

Returns `true` if the bond is an interresidue amide bond; otherwise, it returns `false`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsAmideBond* object is dynamically allocated and owned by the caller.

OEIsAromaticAtom

```
struct OEIsAromaticAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsAromaticAtom* functor that identifies aromatic atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator() *CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsAromatic* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsAromaticAtom* object is dynamically allocated and owned by the caller.

OEIsAromaticBond

```
struct OEIsAromaticBond : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEIsAromaticBond* functor that identifies aromatic bonds (*OEBondBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator() *CreateCopy* *CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator() *CreateCopy*

operator()

```
bool operator()(const OEBondBase& bond) const
```

Returns `true`, if the *OEBondBase.IsAromatic* method returns `true` for the given *OEBondBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsAromaticBond* object is dynamically allocated and owned by the caller.

OEIsAtomHybridization

```
class OEIsAtomHybridization : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsAtomHybridization* functor that identifies atoms with the given hybridization.

See also

- *OEAssignHybridization* function
- *OEHybridization* namespace

Constructors

```
OEIsAtomHybridization(unsigned int hyb)
```

Default constructors that creates an *OEIsAtomHybridization* functor with the given hybridization.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.GetHyb* method returns the same hybridization with which the *OEIsAtomHybridization* object was initialized.

Warning

Prior to using the *OEIsAtomHybridization* functor, the hybridization of a molecule should be perceived by calling the *OEAssignHybridization* function. Otherwise the *OEGetHybridization* function is called every time to perceive the hybridization of the atom on the fly.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsAtomHybridization* object is dynamically allocated and owned by the caller.

OEIsBridgeHead

```
class OEIsBridgeHead : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsBridgeHead* functor that identifies bridge head atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true` if the atom is a bridge head of a ring, otherwise it returns `false`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsBridgeHead* object is dynamically allocated and owned by the caller.

OEIsCAlpha

```
class OEIsCAlpha : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsCAlpha* functor that identifies atoms (*OEAtomBase*) with *OEPDBAtomName.CA* PDB index.

See also

- *Residue Data Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if

- the *OEAtomBase.IsCarbon* method returns `true` and
- the *OEHasResidue* returns `true` and
- the *OEGetPDBAtomIndex* returns *OEPDBAtomName.CA*

for the given *OEAtomBase*, `atom`, object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsCAlpha* object is dynamically allocated and owned by the caller.

OEIsCarbon

```
struct OEIsCarbon : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsCarbon* functor that identifies carbon atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsCarbon* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsCarbon* object is dynamically allocated and owned by the caller.

OEIsChiralAtom

```
struct OEIsChiralAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsChiralAtom* functor that identifies chiral atoms (*OEAtomBase*).

See also

- chiral atom

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsChiral* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsChiralAtom* object is dynamically allocated and owned by the caller.

OEIsChiralBond

```
struct OEIsChiralBond : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEIsChiralBond* functor that identifies chiral bonds (*OEBondBase*).

See also

- chiral bond

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEBondBase& bond) const
```

Returns `true`, if the *OEBondBase.IsChiral* method returns `true` for the given *OEBondBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsChiralBond* object is dynamically allocated and owned by the caller.

OEIsDisulfideBridgeBond

```
class OEIsDisulfideBridgeBond : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents the *OEIsDisulfideBridgeBond* functor that identifies the disulfide bonds.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsDisulfideBridgeBond()
```

Default constructor.

operator()

```
bool operator()(const OEChem::OEBondBase& bond) const
```

Returns `true` if the bond is disulfide; otherwise, it returns `false`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsDisulfideBridgeBond* object is dynamically allocated and owned by the caller.

OEIsHalide

```
struct OEIsHalide : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsHalide* functor that identifies halide (halogen) atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsHalogen* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHalide* object is dynamically allocated and owned by the caller.

OEIsHalogen

```
struct OEIsHalogen : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsHalogen* functor that identifies halogen atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsHalogen* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHalogen* object is dynamically allocated and owned by the caller.

OEIsHeavy

```
struct OEIsHeavy : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsHeavy* functor that identifies heavy atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *GetAtomicNum* method is greater than 1 for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHeavy* object is dynamically allocated and owned by the caller.

See also

- Example program *SizeFilter.cs*
- Example program *MolStats.cs*

OEIsHetero

```
struct OEIsHetero : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsHetero* functor that identifies hetero atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if both the *OEAtomBase.IsCarbon* and the *OEAtomBase.IsHydrogen* methods return `false` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHetero* object is dynamically allocated and owned by the caller.

OEIsHydrogen

```
struct OEIsHydrogen : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsHydrogen* functor that identifies hydrogen atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsHydrogen* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHydrogen* object is dynamically allocated and owned by the caller.

OEIsIntraResidueBond

```
class OEIsIntraResidueBond : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents the *OEIsIntraResidueBond* functor that identifies intra-residue bonds.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsIntraResidueBond()
```

Default constructor.

operator()

```
bool operator()(const OEChem::OEBondBase& bond) const
```

Returns `true` if the bond is intra-residue; otherwise, it returns `false`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntraResidueBond* object is dynamically allocated and owned by the caller.

OEIsInvertibleNitrogen

```
struct OEIsInvertibleNitrogen : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsInvertibleNitrogen* functor that identifies invertible nitrogen atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsNitrogen* method returns `true` for the given *OEAtomBase* object and the atom has a degree of 3, a valence of 3, is not aromatic, and has less than 3 ring bonds.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsInvertibleNitrogen* object is dynamically allocated and owned by the caller.

OEIsMDLStereoGroup

```
class  OEIsMDLStereoGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents *OEIsMDLStereoGroup* functor that identifies groups (*OEGroupBase*) that store MDL enhanced stereo information.

See also

- *OEGroupType* for built-in group types.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator() *CreateCopy* *CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator() *CreateCopy*

Constructors

```
OEIsMDLStereoGroup()
```

Default constructor.

operator()

```
bool operator()(const OEGroupBase& group) const
```

Returns `true`, if the number returned by the *OEGroupBase.GetGroupType* method is equivalent to any of the following constants:

- *OEGroupType.MDLAbsStereo*
- *OEGroupType.MDLAndStereo*
- *OEGroupType.MDLOrStereo*

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsMDLStereoGroup* object is dynamically allocated and owned by the caller.

OEIsMember

```
template<class C>
class OEIsMember : public OESystem::OEUnaryPredicate<C>
```

This class represents *OEIsMember* template functor that identifies all of the objects which are in an STL container.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator() *CreateCopy* *CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator() CreateCopy

Constructors

```
OEIsMember()
```

Default constructors.

```
OEIsMember(std::set<C>& v)
OEIsMember(std::list<C>& v)
OEIsMember(std::deque<C>& v)
OEIsMember(std::vector<C>& v)
```

Constructs the functor using an STL container of type 'C', which contains objects or pointers to objects. Any of the elements of the STL container will be considered to be 'a member' of the functor.

```
OEIsMember(OESystem::OEIter<C>& i)
```

Constructs the functor using an iterator (*OEIter*) over values of the same type as the functor template argument.

```
OEIsMember(OESystem::OEIterBase<C>* i)
```

Constructs the functor using an iterator (*OEIterBase*) over values of the same type as the functor template argument.

```
template<class T>
OEIsMember(T bgn, T end)
```

Constructs the functor using a sequence of values. The values from 'bgn' up to, but not including, 'end' are used to construct the functor. Using this constructor, the functor may be constructed from arrays and STL containers.

operator()

```
bool operator()(const C& arg) const
```

Returns `true` if the object arg (or a pointer to arg) is contained in the STL container passed to the functor on construction.

Add

```
bool Add(C& item)
```

Add a new element to be considered as 'a member' of the functor.

CreateCopy

```
OESystem::OEUnaryPredicate<C>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsMember* object is dynamically allocated and owned by the caller.

Set

```
template<class T>
bool Set(T bgn, T end)
```

All elements specified by the given iterators will be considered to be 'a member' of the functor.

OEIsMemberPtr

```
template<class C>
class OEIsMemberPtr : public OESystem::OEUnaryPredicate<C>
```

This class represents *OEIsMemberPtr* template functor that performs set membership tests. This functor differs from the *OEIsMember* functor in that the internal storage type differs from the test type. While instances of abstract classes cannot be easily stored, however, pointers to abstract class instances do not pose the same storage problems. *OEIsMemberPtr* stores pointers to data types, while the template type of the class is a single dereferenced version of the storage type.

See also

- Example program *MinPath.cs*

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsMemberPtr()
```

Default constructor.

```
OEIsMemberPtr(OESystem::OEIter<C>& i)
```

Constructs the functor using an iterator (*OEIter*) over values of the same type as the functor template argument. Addresses instead of instances or references to the data are stored internally.

```
OEIsMemberPtr(OESystem::OEIterBase<C>* i)
```

Constructs the functor using an iterator (*OEIterBase*) over values of the same type as the functor template argument. Addresses instead of instances or references to the data are stored internally.

```
template<class T>  OEIsMemberPtr(T bgn, T end)
```

Constructs the functor using a sequence of values. The values from 'bgn' up to, but not including 'end' are used to construct the functor. Using this constructor, the functor may be constructed from arrays and STL containers.

operator()

```
bool operator()(const C& arg) const
```

Returns `true` if a pointer to the object 'arg' is contained in the functor.

Add

```
bool Add(const C* item)
```

Add a new pointer to be considered as 'a member' of the functor.

CreateCopy

```
OESystem::OEUnaryPredicate<C>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsMemberPtr* object is dynamically allocated and owned by the caller.

Set

```
template<class T>
bool Set(T bgn, T end)
```

OEIsMetal

```
struct OEIsMetal : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsMetal* functor that identifies metal atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsMetal* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsMetal* object is dynamically allocated and owned by the caller.

OEIsNitrogen

```
struct OEIsNitrogen : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsNitrogen* functor that identifies nitrogen atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsNitrogen* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsNitrogen* object is dynamically allocated and owned by the caller.

OEIsNonRingAtomDoubleBondedToRing

```
struct OEIsNonRingAtomDoubleBondedToRing : public OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase>
```

This class represents the *OEIsNonRingAtomDoubleBondedToRing* functor that identifies non-ring atoms double bonded to a ring.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsInRing* method returns `false` for the given *OEAtomBase* object, and the atom is bonded to an atom where the *OEAtomBase.IsInRing* method returns `true` and the order of the bond as determined by the *OEBondBase.GetOrder* method is 2.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsNonRingAtomDoubleBondedToRing* object is dynamically allocated and owned by the caller.

See also

- Example program *RingSubset.cs*
- Example program *ExtractScaffold.cs*

OEIsomericConfTest

```
class OEIsomericConfTest : public OEConfTestBase
```

This class represents *OEIsomericConfTest*.

This implementation of *OEConfTestBase* combines subsequent connection tables into a multi-conformer molecule if they:

1. Have the same numbers of atoms and bonds in the same order
2. Each atom and bond must have identical properties with its order correspondent in the subsequent connection table
3. Have the same atom and bond stereochemistry
4. Have the same title (optional)

OEIsomericConfTest shouldn't be used for reading multi-conformer molecules in PDB or XYZ file format. These file formats typically require the use of `OEDetermineConnectivity` (and `OEPerceiveBondOrders`) which may potentially perceive and create bonds in different orders (or arbitrarily assign bond orders and Kekulé forms differently) for different conformers of the same molecule. This disrupts the ordering (and property) constraints described above. Generally, file formats other than PDB and XYZ should be used for storing multi-conformer molecules (as consecutive connection tables), but if unavoidable, the *OEAbsCanonicalConfTest* class should be used instead.

The following methods are publicly inherited from *OEConfTestBase*:

CombineMols	*CompareMols*	*CreateCopy*

Constructors

```
OEIsomericConfTest(bool compTitles = true)
```

Constructs an *OEIsomericConfTest* object. The argument 'compTitle' specifies whether to compare molecule titles (accessed by `OEMolBase.GetTitle`). If the constructor is called with no arguments or with the argument `true`, molecule titles will be required to be the same. Otherwise, the titles will not be compared. In the latter instance, each conformer will have the individual title of its original connection table and the multi-conformer molecule will reflect the title of the active conformer.

CombineMols

```
bool CombineMols(OEMCMolBase& m1, OEMolBase& m2)
```

CompareMols

```
bool CompareMols(OEMolBase& m1, OEMolBase& m2) const
```

Returns whether the two *OEMolBase* objects are the same by checking the four conditions listed at the top.

CreateCopy

```
OEConfTestBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsomericConfTest* object is dynamically allocated and owned by the caller.

OEIsOxygen

```
struct OEIsOxygen : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsOxygen* functor that identifies oxygen atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsOxygen* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsOxygen* object is dynamically allocated and owned by the caller.

OEIsPhosphorus

```
struct OEIsPhosphorus : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsPhosphorus* functor that identifies phosphorus atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	CreateCopy	CreatePredicateCopy

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	CreateCopy

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsPhosphorus* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsPhosphorus* object is dynamically allocated and owned by the caller.

OEIsPolar

```
struct OEIsPolar : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsPolar* functor that identifies polar atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	CreateCopy	CreatePredicateCopy

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	CreateCopy

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsPolar* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsPolar* object is dynamically allocated and owned by the caller.

OEIsPolarHydrogen

```
struct OEIsPolarHydrogen : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsPolarHydrogen*.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

OEIsProductGroup

```
class  OEIsProductGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents the *OEIsProductGroup* functor that identifies groups (*OEGroupBase*) that represent reaction component information which contain at least one atom identified by *OEAtomBase.GetRxnRole* as being a *OERxnRole.Product* atom.

See also

- *OEGroupType* for built-in group types.
- *OERxnRole* for reaction role types.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsProductGroup()
```

Default constructor.

operator()

```
bool operator()(const OEGroupBase& group) const
```

Returns `true`, if the value returned by the *OEGroupBase.GetGroupType* method is equal to the following constant:

- *OEGroupType.Component*

and which contain one or more atoms identified as products.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsProductGroup* object is dynamically allocated and owned by the caller.

OEIsReactantGroup

```
class  OEIsReactantGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents the *OEIsReactantGroup* functor that identifies groups (*OEGroupBase*) that represent reaction component information which contain at least one atom identified by *OEAtomBase.GetRxnRole* as being a *OERxnRole.Reactant* atom.

See also

- *OEGroupType* for built-in group types.
- *OERxnRole* for reaction role types.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsReactantGroup()
```

Default constructor.

operator()

```
bool operator()(const OEGroupBase& group) const
```

Returns `true`, if the value returned by the *OEGroupBase.GetGroupType* method is equal to the following constant:

- *OEGroupType.Component*

and which contain one or more atoms identified as reactants.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsReactantGroup* object is dynamically allocated and owned by the caller.

OEIsRGroup

```
class OEIsRGroup : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsRGroup* functor that identifies atoms (*OEAtomBase*) with any or a specific R-group.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsRGroup(unsigned int idx = 0)
```

Constructs the functor with the internal R-group index. With no arguments, this constructs a functor which returns `true` for any atom with a non-zero R-group index. When constructed with a non-zero integer, it constructs a functor which return `true` for atoms which have the same R-group index as the constructor argument.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.GetAtomicNum()* == 0 and the index returned by the *OEAtomBase.GetMapIdx* method for the given *OEAtomBase* object is equivalent to the non-zero index number with which the functor is constructed.

Returns `true`, if the *OEAtomBase.GetAtomicNum()* == 0 and the index returned by the *OEAtomBase.GetMapIdx* method for the given *OEAtomBase* object is non-zero when the index number with which the functor is constructed is `0`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsRGroup* object is dynamically allocated and owned by the caller.

OEIsRotor

```
class OEIsRotor : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEIsRotor* functor that identifies rotatable bonds (*OEBondBase*).

See also

- rotatable bond
- `OEGetRotatableBondCount` function in **OEMolProp TK**
- Example programs *MolStats.cs* and *RotorCount.cs*

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsRotor(bool includeHydrogen = false)
```

Default and copy constructors.

operator()

```
bool operator()(const OEBondBase& bond) const
```

Returns `true`, if the *OEBondBase.IsRotor* method returns `true` for the given *OEBondBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsRotor* object is dynamically allocated and owned by the caller.

OEIsRxnComponentGroup

```
class  OEIsRxnComponentGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents the *OEIsRxnComponentGroup* functor that identifies groups (*OEGroupBase*) that represent reaction component information which contain at least one atom identified by *OEAtomBase.GetRxnRole* as being any of:

- *OERxnRole.Reactant*
- *OERxnRole.Agent*
- *OERxnRole.Catalyst*
- *OERxnRole.Product*

See also

- *OEGroupType* for built-in group types.
- *OERxnRole* for reaction role types.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsRxnComponentGroup()
```

Default constructor.

operator()

```
bool operator()(const OEGroupBase& group) const
```

Returns `true`, if the value returned by the *OEGroupBase.GetGroupType* method is equal to the following constant:

- *OEGroupType.Component*

and which contain one or more atoms identified as either a reactant or product.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsRxnComponentGroup* object is dynamically allocated and owned by the caller.

OEIsSulfur

```
struct OEIsSulfur : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsSulfur* functor that identifies sulfur atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase.IsSulfur* method returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsSulfur* object is dynamically allocated and owned by the caller.

OEIsTermHeavyAtom

```
struct OEIsTermHeavyAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsTermHeavyAtom* functor that identifies terminal heavy atoms (*OEAtomBase*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OEIsHeavy* method returns `true` for the given *OEAtomBase* object and the atom is a terminal atom in the molecule.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsTermHeavyAtom* object is dynamically allocated and owned by the caller.

OEIsValidAtomValence

```
class OEIsValidAtomValence : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsValidAtomValence* functor that identifies valid atoms (*OEAtomBase*) by the **OpenEye valence model**.

See also

- section *OpenEye Valence Model*
- *OECheckAtomValence* function
- *OECheckAtomValences* function
- *OEIsValidMDLAtomValence* functor for **MDL valence model**

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the *OECheckAtomValences* function returns `true` for the given *OEAtomBase* object.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsValidAtomValence* object is dynamically allocated and owned by the caller.

OEIsValidMDLAtomValence

```
class OEIsValidMDLAtomValence : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsValidMDLAtomValence* functor that identifies valid atoms (*OEAtomBase*) by the **MDL valence model**.

See also

- section *MDL Valence Model*
- *OEDefaultMDLHCount* function
- *OEAssignMDLHydrogens* function
- *OEMDLGetValence* function
- *OEIsValidAtomValence* functor for **OpenEye valence model**

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true` if the given *OEAtomBase* object is determined to be valid by MDL valence conventions.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsValidMDLAtomValence* object is dynamically allocated and owned by the caller.

OELibraryGen

```
class OELibraryGen
```

This class represents *OELibraryGen* that is designed to perform library generation.

See also

- *Reactions* chapter
- SMIRKS
- *Library generation* example
- *OEUniMolecularRxn* class

Constructors

```
OELibraryGen()
```

Default constructor.

```
OELibraryGen(const OELibraryGen& rhs)
```

Copy constructor.

```
OELibraryGen(const char * smirks, bool strictSmirks = true)
```

Creates a *OELibraryGen* object with the given SMIRKS reaction. Success of initialization can be tested using the operator *OELibraryGen.operator bool* method.

Passing a value of `false` to the second method argument will relax the strict SMIRKS restrictions. A mapped reactant atom that does not have a corresponding mapped product atom is valid, and will be destroyed as part of the reaction. Unmapped reactant atoms will be used to match the reactant pattern, but are not destroyed when the reaction is applied. The resulting SMIRKS like reactions may therefore be more easily readable by humans as fewer atoms may be required to be mapped.

Atoms in the provided reaction must have unique mapping numbers, otherwise there is ambiguity as to the correspondence of reactant to product atoms. A duplicate mapping number warning message indicates that the SMIRKS must be corrected prior to use.

smirks
The string representing the reaction for library generation.

strictSmirks
The parameter that specifies the interpretation of the SMIRKS semantics.

See also

- *OELibraryGen.Init* method
- SMIRKS

operator()

```
bool operator()(OEMolBase &) const
```

operator bool

```
operator bool() const
```

Returns `true`, if the *OELibraryGen* object was initialized with no failures using either the constructors or one of the *OELibraryGen.Init* methods.

The method will return `false` when the *OELibraryGen* object has not been properly initialized as it will be unable to apply transform operations to molecules.

AddStartingMaterial

```
unsigned int AddStartingMaterial(const OEMolBase& mol,
                                 unsigned int reacnum,
                                 bool umatch = true)
unsigned int AddStartingMaterial(OESystem::OEIter<OEMolBase> &,
                                 unsigned int reacnum,
                                 bool umatch = true)
unsigned int AddStartingMaterial(OESystem::OEIterBase<OEMolBase> *,
                                 unsigned int reacnum,
                                 bool umatch = true)
```

Initializes the starting materials corresponding to a reaction component. An iterator over molecules or a single molecule may be passed as the first argument to the overloaded methods. Subsequent calls to the *OELibraryGen.AddStartingMaterial* methods append to the list of starting materials set in prior calls. The second argument (`reacnum`) specifies the reactant to which the starting materials correspond. The return value is the number of starting materials that were successfully added for the specified reaction component.

A copy of the starting material molecules are stored internally with the atom maps from the reactant pattern. This is done for efficiency, as each product molecule requires very little computational work to be done because of the starting material preprocessing.

The final argument (`umatch`) is used to control the pattern matching of the reactant pattern to the starting material. If the value passed is `true`, only matches that contain a unique set of atoms relative to previously identified matches are used. If the value is `false`, every possible match including those related by symmetry will be used. Reactant patterns are unique matched by default.

See also

- *OELibraryGen.ClearStartingMaterial* method
- *OELibraryGen.SetStartingMaterial* method
- *Starting Materials* section of the :ref:

ClearStartingMaterial

```
void ClearStartingMaterial(unsigned reacnum)
```

Removes any starting materials previously set by either the *OELibraryGen.AddStartingMaterial* or the *OELibraryGen.SetStartingMaterial* methods corresponding to the reactant pattern number (‘reacnum’) passed as the argument to the method.

GetAssignMapIdx

```
bool GetAssignMapIdx() const
```

See also

- Corresponding *OELibraryGen.SetAssignMapIdx* setter method

GetClearCoordinates

```
bool GetClearCoordinates() const
```

See also

- Corresponding *OELibraryGen.SetClearCoordinates* setter method

GetExplicitHydrogens

```
bool GetExplicitHydrogens() const
```

See also

- Corresponding *OELibraryGen.SetExplicitHydrogens* setter method

GetProducts

```
OESystem::OEIterBase<OEMolBase> *GetProducts() const
```

Returns an iterator over all the products generated by the library generation.

GetRemoveUnmappedFragments

```
bool GetRemoveUnmappedFragments() const
```

See also

- Corresponding *OELibraryGen.SetRemoveUnmappedFragments* setter method

GetStartingMaterial

```
OESystem::OEIterBase<OEMolBase> *GetStartingMaterial(unsigned int rnum) const
```

Returns an iterator over all the starting materials that have been successfully added to the *OELibraryGen* object.

See also

- *OELibraryGen.AddStartingMaterial* method
- *OELibraryGen.SetStartingMaterial* method

GetTitleSeparator

```
const std::string& GetTitleSeparator() const
```

See also

- Corresponding *OELibraryGen.SetTitleSeparator* setter method

GetValenceCorrection

```
bool GetValenceCorrection() const
```

See also

- Corresponding *OELibraryGen.SetValenceCorrection* setter method

GetValidateKekule

```
bool GetValidateKekule() const
```

See also

- Corresponding *OELibraryGen.SetValidateKekule* setter method

Init

```
bool Init(const OEQMolBase& mol, bool strictSmirks = true)
bool Init(const char* smirks, bool strictSmirks = true)
```

The first method initializes an *OELibraryGen* object with the *OEQMolBase* object. The second method initializes an *OELibraryGen* object with the 'smirks' pattern. By default, the SMIRKS string is interpreted using strict semantics.

Passing a value of `false` to the second method argument will relax the strict SMIRKS restrictions. A mapped reactant atom that does not have a corresponding mapped product atom is valid, and will be destroyed as part of the reaction. Unmapped reactant atoms will be used to match the reactant pattern, but are not destroyed when the reaction is applied. The resulting SMIRKS like reactions may therefore be more easily readable by humans as fewer atoms may be required to be mapped.

NumReactants

```
unsigned int NumReactants() const
```

Returns the number of separate reactants that the *OELibraryGen* object perceived during initialization. The value returned will be equal to the number of graphically disconnected reactants that are not grouped together using the SMIRKS parts primitive.

NumPossibleProducts

```
unsigned int NumPossibleProducts() const
```

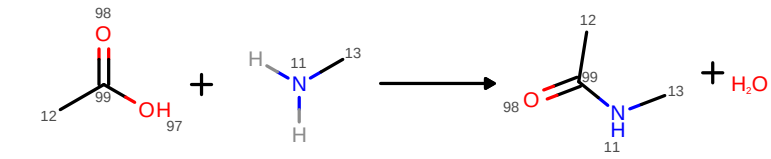
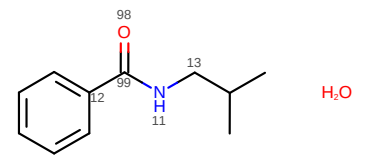
Returns the number of all possible products generated by the library generation. A warning message will be thrown if the number possible combinations exceeds the 2^{32} limit.

SetAssignMapIdx

```
bool SetAssignMapIdx(bool)
```

Sets whether the atom mapping of the reaction is transferred to the product. The default setting is `false` i.e. no mapping will be transferred to the product.

Table 1: **Example of assigning map indices**

reaction	product
	

SetClearCoordinates

```
bool SetClearCoordinates(bool clearcoords)
```

Sets whether the coordinates of a molecule are cleared when the molecule is added to the library generation by calling the *OELibraryGen.AddStartingMaterial* or *OELibraryGen.SetStartingMaterial* methods. The default setting is `false` i.e. the coordinates will be kept.

Note

The *OELibraryGen* modifies the molecular graph of the reactant(s) to generate products. Generating valid 2D or 3D coordinates for the products is beyond the scope of the *OELibraryGen* class.

It is recommended to either clear the coordinates by invoking the *OELibraryGen.SetClearCoordinates* method or regenerate the coordinates of the products afterwards.

SetExplicitHydrogens

```
bool SetExplicitHydrogens(bool explcitH)
```

Sets whether to add explicit hydrogens or suppress them when a starting material is added to the library. The default setting is `true` i.e. explicit hydrogens will be added.

See also

- *Starting Materials* section of the *Reactions* chapter

SetRemoveUnmappedFragments

```
bool SetRemoveUnmappedFragments(bool remove)
```

Sets whether unmapped components of the products are removed or kept prior to returning them by the *OELibraryGen.GetProducts* method. The default setting is `false` i.e. unmapped components will be kept.

See also

- *Atom Mapping* section of the *Reactions* chapter

SetStartingMaterial

```
unsigned int SetStartingMaterial(const OEMolBase& mol,
                                 unsigned int reacnum,
                                 bool umatch = true)
unsigned int SetStartingMaterial(OESystem::OEIter<OEMolBase> &,
                                 unsigned int reacnum,
                                 bool umatch = true)
unsigned int SetStartingMaterial(OESystem::OEIterBase<OEMolBase> *,
                                 unsigned int reacnum,
                                 bool umatch = true)
```

Initializes the starting materials corresponding to a reaction component. An iterator over molecules or a single molecule may be passed as the first argument to the overloaded methods. Subsequent calls to the *OELibraryGen.SetStartingMaterial* methods discard the list of starting materials set in prior calls. The second argument ('reacnum') specifies the reactant to which the starting materials correspond.

A copy of the starting material molecules are stored internally with the atom maps from the reactant pattern. This is done for efficiency, as each product molecule requires very little computational work to be done because of the starting material preprocessing.

The final argument ('umatch') is used to control the pattern matching of the reactant pattern to the starting material. If the value passed is `true`, only matches that contain a unique set of atoms relative to previously identified matches are used. If the value is `false`, every possible match including those related by symmetry will be used. Reactant patterns are unique matched by default.

See also

- *OELibraryGen.ClearStartingMaterial*
- *OELibraryGen.AddStartingMaterial*
- *Starting Materials* section of the *Reactions* chapter

SetTitleSeparator

```
bool SetTitleSeparator(const char* c)
```

Sets the separator or delimiter used by the *OELibraryGen* object to compose product molecule titles. Product molecule titles are created by concatenating reactant molecule titles separated by a title separator or delimiter. The default separator is the underscore character _.

Note

The title separator must be set prior to calling *OELibraryGen.GetProducts*. Calling *OELibraryGen.SetTitleSeparator* after calling *OELibraryGen.GetProducts* will have no affect on the resultant product molecule titles.

See also

- *Reactant Tracking* section of the *Reactions* chapter

SetValenceCorrection

```
bool SetValenceCorrection(bool)
```

Sets whether valence correction is applied to the products prior to returning them by the *OELibraryGen.GetProducts* method. The default setting is `false` i.e. no valence correction will be performed.

See also

- *Valence Correction Modes* section of the *Reactions* chapter

SetValidateKekule

```
bool SetValidateKekule(bool validate)
```

Sets whether the generated products are simply returned after a Kekulization attempt, or if the Kekulé form is validated more rigorously. If the extra validation is enabled and an invalid Kekulé form is identified, alternatives are tried by

adding and removing implicit hydrogens from aromatic nitrogens to produce a valid Kekulé form. No further attempt is made to return the most reasonable or chemically sensible result. By default, the extra Kekulé validation is enabled. The default setting is `true` i.e. Kekulé form validation will be performed.

See also

- *Product Kekulization* section of the *Reactions* chapter

OELingoSim

```
class OELingoSim
```

This class represents *OELingoSim*.

The *OELingoSim* class provides a fast 2D molecular similarity method. The construction of an *OELingoSim* object based on a single molecule or canonical string describing the molecule. Similarity between this original molecule and any subsequent molecule can then be rapidly calculated using another molecule or its canonical string.

See also

- LINGO
- [Grant-2006]
- [Vidal-2005]
- [Vidal-2006]

Constructors

```
OELingoSim(const char* str = 0,
           unsigned type = OELingoType::Default)
OELingoSim(const std::string& str,
           unsigned type = OELingoType::Default)
```

Constructs a *OELingoSim* object with a canonical string. The string API of *OELingoSim* class allows use of any string in both construction and similarity testing (*OELingoSim.Similarity*). While canonical SMILES or canonical isomeric SMILES are the most common string to be used, there is absolutely no requirement to use these strings in generating lingos comparisons. Care must be taken, however, to make sure consistent string types are passed to the constructors and similarity functions! The set of valid types are defined in the *OELingoType* namespace.

```
OELingoSim(const OEMolBase& mol,
           unsigned type = OELingoType::Default)
```

Constructs a *OELingoSim* object with a query molecule (*OEMolBase*). Constructing an *OELingoSim* object with a molecule provides less flexibility because the strings are generated inside the object and thus limited to a specific string type. The set of valid types are defined in the *OELingoType* namespace.

```
OELingoSim(const OELingoSim& rhs)
```

Copy constructor.

GetType

```
unsigned GetType() const
```

Init

```
bool Init(const OEMolBase& mol)
```

(Re)initialize the *OELingoSim* object with a molecule.

```
bool Init(const char* str)
bool Init(const std::string& str)
```

(Re)initialize the *OELingoSim* object with a string.

> **Note**
>
> The implementation type (*OELingoType*), set in the constructor, is unchanged by the *OELingoSim.Init* methods.

IsValid

```
bool IsValid() const
```

Tests for successful initialization of the *OELingoSim* object. If initialization was attempted with an invalid molecule, a string shorter than four characters, or was default constructed it will not be valid. Similarity of any object to an invalid *OELingoSim* will always be `0.0`.

Similarity

```
float Similarity(const char* str) const
float Similarity(const OEMolBase& mol) const
float Similarity(const std::string& cansmi) const
```

Calculates the lingos similarity between the molecule or string used to create the *OELingoSim* object and the molecule or string passed into the *OELingoSim.Similarity* function.

The *OEMolBase* version is provided for convenience, and is suitable for most uses, but is somewhat less efficient than comparing strings directly.

> **Note**
>
> In the string API of the *OELingoSim.Similarity* method, any string can be used (canonical SMILES, canonical isomeric SMILES, IUPAC name), but care must be taken to make sure consistent string types are passed to

OEMatch

```
class OEMatch : public OEMatchBase
```

This class represents *OEMatch*.

The *OEMatch* class is a concrete instance of the *OEMatchBase* abstract base class. The purpose of an *OEMatch* object is to represent a pairwise correspondence between atoms and/or bonds in two different molecules. The correspondences are normally generated as the result of graph matching operations such as subgraph isomorphism determination, maximal common subgraph search, or supergraph search.

The following methods are publicly inherited from *OEMatchBase*:

operator bool	*GetBonds*	*GetTargetBonds*
Clear	*GetPatternAtoms*	*IsValid*
CreateCopy	*GetPatternBonds*	*NumAtoms*
GetAtoms	*GetTargetAtoms*	*NumBonds*

The following methods are publicly inherited from *OERoleSet*:

AddRole	*GetRoles*
ClearRoles	*HasRole*
DeleteRole	*NumRoles*

Constructors

```
OEMatch()
```

Default constructor.

```
OEMatch(const OEMatch &)
OEMatch(const OEMatchBase &)
```

Copy constructor.

```
OEMatch(const std::vector<OEMatchPair<OEAtomBase> > &)
```

Constructors an *OEMatch* object and initializes it by making a copy of the passed vector of *OEMatchPair* objects which contain atom to atom mappings.

```
OEMatch(const std::vector<OEMatchPair<OEBondBase> > &)
```

Constructors an *OEMatch* object and initializes it by making a copy of the passed vector of *OEMatchPair* objects which contain bond to bond mappings.

```
OEMatch(const std::vector<OEMatchPair<OEAtomBase> > &,
        const std::vector<OEMatchPair<OEBondBase> > &)
```

Constructors an *OEMatch* object and initializes it by making a copy of the passed vectors of *OEMatchPair* objects which contain and atom to atom and bond to bond mappings.

```
OEMatch(std::vector<OEMatchPair<OEAtomBase> > *,
        std::vector<OEMatchPair<OEBondBase> > *)
```

AddPair

```
bool AddPair(const OEBondBase *, const OEBondBase *)
```

Adds a pair of bonds (*OEBondBase*) to the *OEMatch* object. Returns `true` if the bonds are successfully added. Returns `false` and throws a warning if the bonds being added do not belong to the same parent molecule as previously added bonds.

```
bool AddPair(const OEAtomBase *, const OEAtomBase *)
```

Adds a pair of atoms (*OEAtomBase*) to the *OEMatch* object. Returns `true` if the atoms are successfully added. Returns `false` and throws a warning if the atoms being added do not belong to the same parent molecule as previously added atoms.

See also

- Example program *SMARTSAlign.cs*

Clear

```
void Clear()
```

Removes all atom and bond of pairs from the *OEMatch* and clears the *OERoleSet*.

CreateCopy

```
OESystem::OEMatchBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned object derived from *OEMatchBase* is dynamically allocated and owned by the caller.

GetAtoms

```
OESystem::OEIterBase<OEMatchPair<OEAtomBase> >* GetAtoms() const
```

Returns an iterator over *OEMatchPairs* which contain the atom to atom correspondences in the *OEMatchBase* object. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

GetBonds

```
OESystem::OEIterBase<OEMatchPair<OEBondBase> >* GetBonds() const
```

Returns an iterator over *OEMatchPairs* which contain the bond to bond correspondences in the *OEMatchBase* object. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

NumAtoms

```
unsigned int NumAtoms() const
```

Returns the number of atom equivalences in the *OEMatch* object.

NumBonds

```
unsigned int NumBonds() const
```

Returns the number of bond equivalences in the *OEMatch* object.

OEMatchBase

```
class OEMatchBase : public OESystem::OERoleSet
```

The *OEMatchBase* class is the abstract interface for representing the equivalences or correspondences found in a substructure (or similar) match. The *OEMatchBase* class can represent substructure hits, maximal common subgraph fragments, tautomer equivalences, automorphisms and sequence alignments.

The following methods are publicly inherited from *OERoleSet*:

AddRole	*GetRoles*
ClearRoles	*HasRole*
DeleteRole	*NumRoles*

The following classes derive from this class:

- *OEMatch*

operator bool

```
operator bool() const
```

Same as the *OEMatchBase.IsValid* method.

Clear

```
void Clear()=0
```

Removes all atom and bond of pairs from any object derived from *OEMatchBase* and clears the *OERoleSet*.

CreateCopy

```
OEMatchBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned object derived from *OEMatchBase* is dynamically allocated and owned by the caller.

GetAtoms

```
OESystem::OEIterBase<OEMatchPair<OEAtomBase> >* GetAtoms() const = 0
```

Returns an iterator over *OEMatchPairs* which contain the atom to atom correspondences in the *OEMatchBase* object. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

GetBonds

```
OESystem::OEIterBase<OEMatchPair<OEBondBase> >* GetBonds() const = 0
```

Returns an iterator over *OEMatchPairs* which contain the bond to bond correspondences in the *OEMatchBase* object. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

GetPatternAtoms

```
OESystem::OEIterBase<OEAtomBase>* GetPatternAtoms() const
```

Returns an iterator over the match-ordered atoms of the pattern molecule. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

GetPatternBonds

```
OESystem::OEIterBase<OEBondBase>* GetPatternBonds() const
```

Returns an iterator over the match-ordered bonds of the pattern molecule. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

GetTargetAtoms

```
OESystem::OEIterBase<OEAtomBase>* GetTargetAtoms() const
```

Returns an iterator over the match-ordered atoms of the target molecule. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

GetTargetBonds

```
OESystem::OEIterBase<OEBondBase>* GetTargetBonds() const
```

Returns an iterator over the match-ordered bonds of the target molecule. The returned *OEIterBase* pointer should be assigned to an *OEIter* object to prevent memory leaks.

IsValid

```
bool IsValid() const
```

Returns if at least one atom or one bond equivalence is stored in the *OEMatchBase* object.

NumAtoms

```
unsigned int NumAtoms() const = 0
```

Returns the number of atom equivalences in a match.

NumBonds

```
unsigned int NumBonds() const = 0
```

Returns the number of bond equivalences in a match.

OEMatchFunc

```
template<class Arg>
class OEMatchFunc : public OESystem::OEUnaryPredicate<Arg>
```

This class represents *OEMatchFunc* template functor that identifies objects (*OEAtomBase*, *OEBondBase*) that match a query substructure specified in construction.

See also

- *Functor substructure-based matching* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following specializations exist for this template:

Specialized Functor	Definition
OEMatchAtom	*OEMatchFunc* <*OEAtomBase*>
OEMatchGraphMol	*OEMatchFunc* <*OEGraphMol*>
OEMatchMolBase	*OEMatchFunc* <*OEMolBase*>

Constructors

```
OEMatchFunc(const char* smarts)
```

Constructs the functor with the internal substructure search specified by the 'smarts' (SMARTS) pattern.

```
OEMatchFunc(const OEQMolBase& qmol)
```

Constructs the functor with the internal substructure search specified by the 'qmol' query molecule.

```
OEMatchFunc(const OEMolBase& mol, unsigned int aexpr,
            unsigned int bexpr)
```

Constructs the functor with the internal substructure search that is created from 'mol' using the specified expression options. The atom and bond expression options passed as the second and third arguments to the method are defined in the *OEExprOpts* namespace. The expression options are used to convert the atom and bond data into expression trees which are used during the subgraph search.

operator()

```
bool operator()(const Arg& arg) const
```

Returns `true` if the 'arg' object matched by the query molecule defined in construction.

operator bool

```
operator bool()
```

Returns `true`, if the substructure search object was initialized correctly.

CreateCopy

```
OESystem::OEUnaryFunction<Arg, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEMatchFunc* object is dynamically allocated and owned by the caller.

OEMatchPair

```
template<class T>
class OEMatchPair
```

This class represents *OEMatchPair*.

The *OEMatchPair* template class represents both atom-atom and bond-bond equivalences in pattern matching. Each *OEMatchPair* represents a correspondence (or matching) between a pattern atom or bond in the query molecule, and a target atom or bond in the target molecule, respectively.

Constructors

```
OEMatchPair()
```

Default constructor.

```
OEMatchPair(const T* p, const T* t)
```

Initializes the *OEMatchPair* object with either an atom (*OEAtomBase*) or a bond (*OEBondBase*) pair.

OEMCMolBase

```
class OEMCMolBase : public OEMolBase
```

The *OEMCMolBase* class provides the basic multi-conformer molecule in *OEChem*. It is an abstract base class which defines the interface for multi-conformer molecule implementations. Coordinates can be stored in *OEHalfFloat* (16-bit), `float` (32-bit), `double` (64-bit), or `long double` (>= 64-bit). The precision is determined by the constructor argument given to *OEMol*, taken from the *OEMCMolType* namespace. *OEMCMolBase* have an interface which allow access to conformers in a modal (*OEMCMolBase.GetActive*) or a non-modal manner (*OEMCMolBase.GetConfs*).

The following methods are publicly inherited from *OEMolBase*:

operator=	*GetDimension*	*NumBonds*
operator bool	*GetEnergy*	*NumGroups*
Clear	*GetGroup*	*OrderAtoms*
ClearCoords	*GetGroups*	*OrderBonds*
Compress	*GetMaxAtomIdx*	*ResetPerceived*
Count	*GetMaxBondIdx*	*SetCoords*
CreateCopy	*GetTitle*	*SetDimension*
DeleteAtom	*HasPerceived*	*SetEnergy*
DeleteBond	*IsDataType*	*SetPerceived*
DeleteGroup	*IsDeleted*	*SetRxn*
GetAtom	*IsRxn*	*SetTitle*
GetAtoms	*NewAtom*	*Sweep*
GetBond	*NewBond*	*UnCompress*
GetBonds	*NewGroup*	
GetCoords	*NumAtoms*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

ClearBase

```
void ClearBase()=0
```

Clear the generic data from the *OEBase* base class of this object. Equivalent to just calling *OEBase.Clear* without actually clearing away molecule data like atoms, bonds, and conformers.

ClearMCMol

```
void ClearMCMol()=0
```

Clears molecule data like atoms, bonds, and conformers without clearing away the *OEBase* generic data.

ClearMolBase

```
void ClearMolBase()=0
```

Equivalent to calling *OEMolBase.Clear*.

DeleteConf

```
bool DeleteConf(OEConfBase *)=0
```

Deletes the conformer which is passed in from the *OEMCMolBase* object.

DeleteConfs

```
void DeleteConfs()=0
```

> **Warning**
>
> *OEMCMolBase.DeleteConfs* leaves the *OEMCMolBase* object in an unstable state. It should be used with care and followed shortly with a call to *OEMCMolBase.NewConf*.

Deletes all of the conformers from the *OEMCMolBase* object. This is a very useful function for creating a new transformed *OEMCMolBase* from an untransformed molecule.

> **See also**
>
> *Listing 3* code example in the *Conformer Creation* section

GetActive

```
OEConfBase* GetActive() const = 0
```

Returns the currently active conformer of the *OEMCMolBase* object.

> **Note**
>
> The *OEMCMolBase.GetActive* and *OEMCMolBase.SetActive* methods are often sufficient for accessing conformations in multi-conformer molecules.

> **See also**
>
> - *OEMCMolBase.PopActive* method
> - *OEMCMolBase.PushActive* methods
> - *OEMCMolBase.SetActive* method

GetConf

```
OEConfBase* GetConf(const OESystem::OEUnaryPredicate<OEChem::OEConfBase>& pred) const = 0
```

Returns the first conformer in the molecule for which the predicate passed in returns `true`.

GetConfs

```
OESystem::OEIterBase<OEConfBase>* GetConfs() const = 0
OESystem::OEIterBase<OEConfBase>* GetConfs(const OESystem::OEUnaryPredicate
↪<OEChem::OEConfBase>& pred) const = 0
```

Returns an iterator over the conformers in the multi-conformer molecule. The return value of this function should always be assigned to an *OEIter* object. The function which takes no arguments returns an iterator over all of the conformers. The function which takes a predicate returns an iterator which only contains conformers for which the predicate returns `true`.

GetMaxConfIdx

```
unsigned int GetMaxConfIdx() const = 0
```

For performance reasons, conformer indices are not guaranteed to be contiguous. *OEMCMolBase.GetMaxConfIdx* returns a lower bound for the maximum conformer index i.e. all indices returned by *OEConfBase.GetIdx* of the multi-conformer molecule will be less than this value. The value returned by *OEMCMolBase.GetMaxConfIdx* is always at least as large as *OEMCMolBase.NumConfs*.

This method is useful for allocating temporary external data structures that need to be indexed by conformer index, as returned by *OEConfBase.GetIdx*.

GetMCMolTitle

```
const char* GetMCMolTitle() const = 0
```

Return the title for the parent molecule, don't fall back to a conformer title like *OEConfBase.GetTitle*.

IsDeleted

```
bool IsDeleted(OEConfBase *) const = 0
```

Returns whether the passed in conformer has already been deleted.

See also

- *OEMolBase.IsDeleted* methods inherited from the *OEMolBase* class

NewConf

```
OEConfBase* NewConf()=0
OEConfBase* NewConf(const OEPlatform::OEHalfFloat *)=0
OEConfBase* NewConf(const float *)=0
OEConfBase* NewConf(const double *)=0
```

(continues on next page)

(continued from previous page)

```
OEConfBase* NewConf(const long double *)=0
OEConfBase* NewConf(const OEMolBase *)=0
OEConfBase* NewConf(const OEConfBase *)=0
OEConfBase* NewConf(const OEConfBase *, const OETrans &)=0
OEConfBase* NewConf(const OEConfBase *, std::vector<OETorsion>& t)=0
```

These methods generate a new conformer that is owned by the current *OEMCMolBase*. Each of the methods will return a pointer to the newly created conformer. The *OEMCMolBase.NewConf* methods act as virtual constructors of the *OEConfBase* objects. *OEMCMolBase.NewConf* constructs a conformer with its default constructor. `NewConf(const OEMolBase *)` and `NewConf(const OEConfBase *)` copy construct a new conformer with the coordinates from the object passed into the function. The objects passed in must have the same graph as the current *OEMCMolBase*. `NewConf` that takes a *OEHalfFloat*, `float`, `double`, or `long double` pointer constructs a new conformer with the coordinates passed in as coords. The array must be of length *GetMaxAtomIdx()* `* 3`, and the coordinates for each atom in the new conformer should be the dimension values in the array starting at `coords[atom->GetIdx() * 3]`.

Passing a `NULL` to any of these methods will effectively do nothing and just return a `NULL` *OEConfBase* pointer.

Warning

The dimension of the conformer will be set to `0` for the NewConf default constructor. The NewConf methods that create conformers from coordinates will set the dimension of the conformer to `3`. The NewConf methods that copy construct will copy the dimension from the source.

NumConfs

```
unsigned int NumConfs() const = 0
```

Returns the number of conformers contained in the *OEMCMolBase* object.

OrderConfs

```
bool OrderConfs(const std::vector<OEConfBase*>& vc)=0
```

Reorders the conformers in the molecule to the order specified in the vector argument. If the vector contains an incomplete list, the remaining conformers will come at the end. This function call changes the order in which the conformers are returned by *OEMCMolBase.GetConfs*, but does not change the conformer indices.

PopActive

```
void PopActive()=0
```

The *OEMCMolBase.PopActive* method along with the *OEMCMolBase.PushActive* method allow to maintain a stack of active conformers.

The *OEMCMolBase.PopActive* method removes the top active conformer from the active stack and makes the next highest conformer in the stack active.

See also

- *OEMCMolBase.GetActive* method

- *OEMCMolBase.SetActive* method
- *OEMCMolBase.PushActive* method

PushActive

```
bool PushActive(OEConfBase *)=0
```

The *OEMCMolBase.PushActive* method along with the *OEMCMolBase.PopActive* method allow to maintain a stack of active conformers.

The *OEMCMolBase.PushActive* method makes the new conformer the active one and pushes the previous active conformer down the stack.

See also

- *OEMCMolBase.GetActive* method
- *OEMCMolBase.SetActive* method
- *OEMCMolBase.PopActive* method

SetActive

```
bool SetActive(OEConfBase *)=0
```

Makes the conformer passed in become the active conformer. The conformer passed in must already be a member of the *OEMCMolBase* object.

Note

The *OEMCMolBase.GetActive* and *OEMCMolBase.SetActive* methods are often sufficient for accessing conformations in multi-conformer molecules.

See also

- *OEMCMolBase.GetActive* method
- *OEMCMolBase.PopActive* method
- *OEMCMolBase.PushActive* method

SweepConfs

```
bool SweepConfs()=0
```

Cleans up unused memory and objects which may be associated with the conformers of the *OEMCMolBase*. Renumber the conformer indices sequentially. This function invalidates the conformer indices of all conformers in a molecule. Note that this function doesn't guarantee that all conformer indices are sequential upon completion (some molecule implementations may treat *OEMolBase.Sweep* as a no-op).

OEMCSFunc

```
class OEMCSFunc
```

The *OEMCSFunc* is an abstract base class that defines the API used for scoring subgraph matches.

The scores generated by implementations of *OEMCSFunc* influence the sorting and retention of maximum common subgraph matches generated by the *OEMCSSearch* and the *OECliqueSearch* classes.

The following classes derive from this class:

- *OEMCSMaxAtoms*
- *OEMCSMaxAtomsCompleteCycles*
- *OEMCSMaxBonds*
- *OEMCSMaxBondsCompleteCycles*

See also

- *OEMCSSearch.SetMCSFunc* method
- *OECliqueSearch.SetMCSFunc* method

Note

Custom implementation of *OEMCSFunc* can be done by deriving from the *OEMCSFunc* class and implementing all of the methods listed below.

operator()

```
double operator()(const OEMolBase& pattern, const OEMolBase& target, OEAtomBase** amap,␣
↪OEBondBase** bmap)=0
```

This method is called automatically by the *OEMCSSearch* and *OECliqueSearch* classes when a common subgraph of two molecules is identified.

The 'pattern' (query) molecule and 'target' molecule currently being matched are passed as the first and second arguments to the function. The arrays of pointers to atoms ('amap') and bonds ('bmap') hold the atom and bond correspondences between the pattern and target. The arrays are the length of the maximum atom and bond indices of the pattern molecule. The indices of the atoms and bonds in the pattern molecule can be used to look up the corresponding atoms and bonds in the target molecule. Subgraphs may not include all pattern atoms. Array positions for unmatched atoms and bonds are assigned to the NULL pointer.

The integer part of the floating point value returned by the method is used to determine maximal common subgraphs.

All integer part scores which are smaller than the maximum computed value for any subgraph are discarded by *OEMCSSearch*. In case of *OECliqueSearch* the any subgraph are discarded that has a smaller scoring value than the maximum score minus the 'range' set by *OECliqueSearch.SetSaveRange*.

The decimal part of the floating point value returned by the method is used to sort the matches found by *OEMCSSearch* or *OECliqueSearch* For example, by scoring matches using the function

$num.\ of\ mapped\ atoms + \frac{num.\ of\ mapped\ bonds}{100}$

all matches which have the same number of subgraph atoms would be retained but the matches would be returned in order of decreasing number of bonds matched.

CreateCopy

```
OEMCSFunc* CreateCopy() const = 0
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEMCSFunc* object is dynamically allocated and owned by the caller.

OEMCSFuncParameter

```
class OEMCSFuncParameter : public OESystem::OEMultiParameter<OEChem::OEMCSFunc>
```

OEMCSFuncParameter represents a parameter that has a value of type *OEMCSFunc*.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OEMultiParameter*:

- *AddLegalEntry*
- *GetDefault* and *SetDefault*
- *GetSetting*
- *GetValue* and *SetValue*

Constructors

```
OEMCSFuncParameter()
OEMCSFuncParameter(const std::string& name)
OEMCSFuncParameter(const OEMCSFuncParameter&)
```

Default and copy constructors.

Constructs an OEMCSFuncParameter instance using the specified set of parameters.

OEMCSMaxAtoms

```
class OEMCSMaxAtoms : public OEMCSFunc
```

This class represents *OEMCSMaxAtoms*.

The *OEMCSMaxAtoms* class is an implementation of *OEMCSFunc* designed to order maximum common substructure matches by the maximum number of atoms included in the graph match. If two common structure matches having the same number of atoms, ties are split based on the number of bonds contained in the match.

See also

- *MCS scoring functions* section
- *OEMCSSearch.SetMCSFunc* method
- *OECliqueSearch.SetMCSFunc* method

The following methods are publicly inherited from *OEMCSFunc*:

operator()	*CreateCopy*

operator()

```
double operator()(const OEMolBase &, const OEMolBase &, OEAtomBase **, OEBondBase **)
```

This method is called by its parent *OEMCSSearch* or *OECliqueSearch* object. The method is called with the pattern molecule, target molecule, and arrays and atoms and bonds containing the correspondences found for the a common structure match. The method computes a value based on the number of atoms and bonds in the common structure match which is used to determine the maximum common structure match.

Scoring function:

$num.\ of\ mapped\ atoms + \frac{num.\ of\ mapped\ bonds}{100}$

CreateCopy

```
OEMCSFunc * CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEMCSMaxAtoms* object is dynamically allocated and owned by the caller.

OEMCSMaxAtomsCompleteCycles

```
class OEMCSMaxAtomsCompleteCycles : public OEMCSFunc
```

This class represents *OEMCSMaxAtomsCompleteCycles*.

The *OEMCSMaxAtomsCompleteCycles* class is an implementation of *OEMCSFunc* designed to order maximum substructure matches by the maximum number of atoms included in the graph match, with a variable priority given to matches which contain complete cycles common to both pattern and target.

See also

- *MCS scoring functions* section
- *OEMCSSearch.SetMCSFunc* method
- *OECliqueSearch.SetMCSFunc* method

The following methods are publicly inherited from *OEMCSFunc*:

operator()	*CreateCopy*

Constructors

```
OEMCSMaxAtomsCompleteCycles(double a = 1.0)
```

The constructor takes a 'penalty' argument, with a default value of `1.0`. The penalty (or scaling) factor is used to subtract from the total score composed of matching atoms and bonds the number of pattern ring bonds that are not matches by a target bond (see scoring function below). This has the affect of scoring matches more highly when pattern ring atoms are matched.

operator()

```
double operator()(const OEMolBase &, const OEMolBase &, OEAtomBase **, OEBondBase **)
```

This method is called by its parent *OEMCSSearch* or *OECliqueSearch* object. The method is called with the pattern molecule, target molecule, and arrays and atoms and bonds containing the correspondences found for the a common structure match. The method computes a value based on the number of atoms and bonds in the common structure match which is used to determine the maximum common structure match.

Scoring function:

$$num.\ of\ mapped\ atoms + \frac{num.\ of\ mapped\ bonds}{100} - penalty \times num.\ of\ unmapped\ cyclic\ query\ bonds$$

The default penalty for each unmapped cyclic query bond is `1.0`.

CreateCopy

```
OEMCSFunc * CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEMCSMaxAtomsCompleteCycles* object is dynamically allocated and owned by the caller.

OEMCSMaxBonds

```
class OEMCSMaxBonds : public OEMCSFunc
```

This class represents *OEMCSMaxBonds*.

The *OEMCSMaxBonds* class is an implementation of *OEMCSFunc* designed to order maximum common substructure matches by the maximum number of bonds included in the graph match. If two common structure matches have the same number of bonds, ties are split based on the number of atoms contained in the match.

See also

- *MCS scoring functions* section
- *OEMCSSearch.SetMCSFunc* method
- *OECliqueSearch.SetMCSFunc* method

The following methods are publicly inherited from *OEMCSFunc*:

operator()	*CreateCopy*

operator()

```
double operator()(const OEMolBase &, const OEMolBase &, OEAtomBase **, OEBondBase **)
```

This method is called by its parent *OEMCSSearch* or *OECliqueSearch* object. The method is called with the pattern molecule, target molecule, and arrays and atoms and bonds containing the correspondences found for the a common structure match. The method computes a value based on the number of atoms and bonds in the common structure match which is used to determine the maximum common structure match.

Scoring function:

$num.\ of\ mapped\ bonds + \frac{num.\ of\ mapped\ atoms}{100}$

CreateCopy

```
OEMCSFunc * CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEMCSMaxBonds* object is dynamically allocated and owned by the caller.

OEMCSMaxBondsCompleteCycles

```
class OEMCSMaxBondsCompleteCycles : public OEMCSFunc
```

This class represents *OEMCSMaxBondsCompleteCycles*.

The *OEMCSMaxBondsCompleteCycles* class is an implementation of *OEMCSFunc* designed to order maximum substructure matches by the maximum number of bonds included in the graph match, with a variable priority given to matches which contain complete cycles common to both pattern and target.

See also

- *MCS scoring functions* section
- *OEMCSSearch.SetMCSFunc* method
- *OECliqueSearch.SetMCSFunc* method

The following methods are publicly inherited from *OEMCSFunc*:

operator()	*CreateCopy*

Constructors

```
OEMCSMaxBondsCompleteCycles(double a = 1.0)
```

The constructor takes a 'penalty' argument, with a default value of `1.0`. The penalty (or scaling) factor is used to subtract from the total score composed of matching atoms and bonds the number of pattern ring bonds that are not matches by a target bond (see scoring function below). This has the affect of scoring matches more highly when pattern ring bonds are matched.

operator()

```
double operator()(const OEMolBase &, const OEMolBase &, OEAtomBase **, OEBondBase **)
```

This method is called by its parent *OEMCSSearch* or *OECliqueSearch* object. The method is called with the pattern molecule, target molecule, and arrays and atoms and bonds containing the correspondences found for the a common structure match. The method computes a value based on the number of atoms and bonds in the common structure match which is used to determine the maximum common structure match.

Scoring function:

$$num.\ of\ mapped\ bonds + \frac{num.\ of\ mapped\ atoms}{100} - penalty \times num.\ of\ unmapped\ cyclic\ query\ bonds$$

The default penalty for each unmapped cyclic query bond is `1.0`.

CreateCopy

```
OEMCSFunc * CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEMCSMaxBondsCompleteCycles* object is dynamically allocated and owned by the caller.

OEMCSSearch

```
class OEMCSSearch
```

This class represents *OEMCSSearch*.

This *OEMCSSearch* class identifies a set of maximum substructures common to two molecules.

See also

- *Maximum Common Substructure Search* section
- Example program *MCS3DAlign.cs*

Constructors

```
OEMCSSearch(unsigned int type = OEMCSType::Default)
```

Default constructor.

```
OEMCSSearch(const OEMCSSearch& rhs)
```

Copy constructor.

```
OEMCSSearch(const char* smarts,
            unsigned int type = OEMCSType::Default)
```

Constructs an *OEMCSSearch* object using a 'smarts' pattern. The SMARTS pattern is parsed into a query molecule (*OEQMolBase*). The valid types of the *OEMCSSearch* are defined in the *OEMCSType* namespace.

```
OEMCSSearch(const OEQMolBase& qmol, bool perception = true,
            unsigned int type = OEMCSType::Default)
```

Constructs an *OEMCSSearch* object using a query molecule (*OEQMolBase*).

qmol
: The query molecule of the MCS search.

type
: The type of the *OEMCSSearch* from the *OEMCSType* namespace.

perception
: If `true`, then ring perception is performed on the query molecule.

```
OEMCSSearch(const OEMolBase &, unsigned int atomexpr,
            unsigned int bondexpr,
            unsigned int type = OEMCSType::Default)
```

Constructs an *OEMCSSearch* object using a molecule and expression options. The atom and bond expression options passed as the second and third arguments to the method are defined in the *OEExprOpts* namespace. The expression options are used to convert the atom and bond data into expression trees which are used during the common subgraph search.

operator bool

```
operator bool() const
```

Test for successful initialization of an *OEMCSSearch* object. If initialization was attempted with an invalid SMARTS pattern or query molecule (*OEQMolBase*) then this method will return `false`. If initialization was completed successfully this method will return `true`. An *OEMCSSearch* object is considered to be uninitialized when constructed with the default constructor.

AddConstraint

```
bool AddConstraint(const OEMatchPair<OEBondBase> &)
bool AddConstraint(const OEMatchPair<OEAtomBase> &)
```

The search space of a maximum common subgraph determination can be restricted by constraining pairs of nodes or edges (atoms or bonds) to be mapped onto one another in all subgraph solutions. All subgraph solutions will contain all constraints specified by this method.

Both *OEMCSSearch.AddConstraint* methods return `true` if a constraint is added successfully. If the pattern atom or bond in the *OEMatchPair* does not exist as part of the query molecule created in the initialization of the *OEMCSSearch* object then *OEMCSSearch.AddConstraint* will return `false`.

Multiple calls to *OEMCSSearch.AddConstraint* using the same pattern atom or bond will return `false` as it is impossible to satisfy multiple simultaneous constraints for a single pattern atom or bond, hence the exclusivity.

ClearConstraints

```
void ClearConstraints()
```

Clears all prior match constraints set with the *OEMCSSearch.AddConstraint* methods.

GetMaxMatches

```
unsigned int GetMaxMatches() const
```

Returns the maximum number of common subgraphs that the *OEMCSSearch* will identify before terminating the search. A value of zero indicates that no arbitrary limit has been set on the total number of subgraphs to be identified. By default, the maximum number of matches is set to `1024` This value can be set using the *OEMCSSearch.SetMaxMatches* method.

GetMinAtoms

```
unsigned int GetMinAtoms() const
```

Returns the minimum number of atoms of a subgraph that can be returned as a maximum common subgraph. This value can be set using the *OEMCSSearch.SetMinAtoms* method.

GetPattern

```
const OEQMolBase& GetPattern() const
```

Returns a read-only reference to the query molecule (*OEQMolBase*) of *OEMCSSearch*. Const *OEQMolBase* methods can be used on the returned *OEQMolBase* reference. If the *OEMCSSearch* object has not been initialized, a reference to an empty molecule will be returned.

Init

```
bool Init(const char* smarts)
```

(Re)initializes an *OEMCSSearch* object using a 'smarts' pattern. The SMARTS pattern is parsed to create a query molecule available for maximum common subgraph matching. The method will return `true` if initialization completes successfully, and `false` upon failure.

Note

Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

```
bool Init(const OEQMolBase& qmol, bool perception = true)
```

(Re)initializes an *OEMCSSearch* object using a query molecule (*OEQMolBase*) reference. The method will return `true` if initialization completes successfully, and `false` upon failure.

qmol
The query molecule of the MCS search.

perception
If `true`, then ring perception is performed on the query molecule.

Note

Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

```
bool Init(const OEMolBase& mol, unsigned int atomexpr,
          unsigned int bondexpr)
```

(Re)initializes an *OEMCSSearch* object using a molecule and expression options that are used to convert the given molecule into a *OEQMolBase* query object.

mol
The *query* molecule of the maximum common substructure search.

atomexpr
This value has to be any combination of atom expressions of the *OEExprOpts* namespace.

bondexpr
This value has to be any combination of bond expressions of the *OEExprOpts* namespace.

Expression options are used to convert the atom and bond data into expression trees which are used during the common subgraph search. The method will return `true` if initialization completes successfully, and `false` upon failure.

Note

Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

See also

- *OEExprOpts* namespace

Match

```
OESystem::OEIterBase<OEMatchBase>* Match(const OEMolBase &,
                                         bool uniquematch = false) const
```

Performs search in order to identify maximum common subgraphs shared between the passed *OEMolBase* and the query molecule which the *OEMCSSearch* object was initialized. It returns an iterator pointer (*OEIterBase*) over graph matches (*OEMatchBase*), and should be assigned to an *OEIter* < *OEMatchBase* > in order to prevent memory leaks.

If the 'uniquematch' argument passed to the methods is `true` then only the unique matches will be returned in the iterator over the matches. By definition, a match or subgraph is considered unique if it differs from all other subgraphs found previously by at least one atom or bond. Additionally, it is also considered unique if the query subgraph is mapped to a different part of the target (See example in the *Maximum Common Substructure Search* section)

SetMCSFunc

```
bool SetMCSFunc(const OEChem::OEMCSFunc &)
```

Stores a copy of the passed *OEMCSFunc* functor in the *OEMCSSearch* object. The functor is then used to evaluate and order subgraphs found during a maximum common subgraph search.

See also

- *OEMCSFunc*
- *OEMCSMaxAtoms*
- *OEMCSMaxBonds*
- *OEMCSMaxAtomsCompleteCycles*
- *OEMCSMaxBondsCompleteCycles*

SetMaxMatches

```
bool SetMaxMatches(unsigned int)
```

Sets the maximum number of maximum common subgraph matches that will be returned by the *OEMCSSearch.Match* method.

Note

The search for maximum common substructures will not terminate immediately upon reaching this limit. The maximum common subgraph cannot be known unless the MCS is composed of all atoms and bonds of at least one of the graphs being compared. The limit of subgraphs to be returned may be reached with a smaller subgraph than the maximum. In such a case the search continues for larger subgraphs until the search is exhausted. *OEMCSSearch.Match* will return the first N maximum common subgraphs where N is less than or equal to the maximum match limit.

The default limit set upon construction of an *OEMCSSearch* object is `1024` matches. This value can be retrieved using the *OEMCSSearch.GetMaxMatches* method.

SetMinAtoms

```
bool SetMinAtoms(unsigned int)
```

Sets the minimum number of atoms required of a subgraph match to be returned a solution by a MCS search. A single atom can be a perfectly valid maximum common subgraph, however, for many applications such a small subgraph

may not be considered useful. Setting the minimum number of atoms to an appropriate size prevents unproductive subgraph matches from being returned by the *OEMCSSearch.Match* method. The default value for the minimum number of atoms in a subgraph match is set to 1 upon construction of an *OEMCSSearch* object. This value can be retrieved using the *OEMCSSearch.GetMinAtoms* method.

OEMMFFTypeResult

```
class OEMMFFTypeResult
```

Used by the *OEMMFFAtomType* function to retrieve the MMFF94 type of an atom.

The following table describes how the result should be interpreted:

GetMMFFType()	GetFailed()	interpretation
OEMMFFType.UNK	true	MMFF atom type **can not** be identified
not OEMMFFType.UNK	true	an **alternative** MMFF atom type is identified
not OEMMFFType.UNK	false	MMFF atom type is identified **successfully**

Constructors

```
OEMMFFTypeResult(unsigned int type, bool failed)
```

Constructor

GetFailed

```
bool GetFailed() const
```

Returns whether MMFF94 type can be assigned to an atom.

GetMMFFType

```
unsigned int GetMMFFType() const
```

Returns MMFF94 type from the *OEMMFFType* namespace.

OEMol

```
class OEMol
```

The *OEMol* class is a specific implementation of multi-conformer molecules.

The following methods are publicly inherited from *OEMolBase*:

operator=	*GetDimension*	*NumBonds*
operator bool	*GetEnergy*	*NumGroups*
Clear	*GetGroup*	*OrderAtoms*
ClearCoords	*GetGroups*	*OrderBonds*
Compress	*GetMaxAtomIdx*	*ResetPerceived*
Count	*GetMaxBondIdx*	*SetCoords*
CreateCopy	*GetTitle*	*SetDimension*
DeleteAtom	*HasPerceived*	*SetEnergy*
DeleteBond	*IsDataType*	*SetPerceived*
DeleteGroup	*IsDeleted*	*SetRxn*
GetAtom	*IsRxn*	*SetTitle*
GetAtoms	*NewAtom*	*Sweep*
GetBond	*NewBond*	*UnCompress*
GetBonds	*NewGroup*	
GetCoords	*NumAtoms*	

The following methods are publicly inherited from *OEMCMolBase*:

DeleteConf	*GetMCMolTitle*	*PushActive*
DeleteConfs	*IsDeleted*	*SetActive*
GetActive	*NewConf*	*SweepConfs*
GetConf	*NumConfs*	
GetConfs	*OrderConfs*	
GetMaxConfIdx	*PopActive*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEMol(const OEMol &)
OEMol(const OEMCMolBase &)
OEMol(const OEMol &, unsigned int type)
OEMol(unsigned int type = OEMolType::OEDefault)
OEMol(const OEMCMolBase &, unsigned int type)
OEMol(const OEQMol &, unsigned int type = OEMCMolType::OEDefault)
OEMol(const OEMolBase &, unsigned int type = OEMCMolType::OEDefault)
OEMol(const OEQMolBase &,
      unsigned int type = OEMCMolType::OEDefault)
OEMol(const OEGraphMol &,
      unsigned int type = OEMCMolType::OEDefault)
```

(continues on next page)

(continued from previous page)

```
OEMol(const OESystem::OEBase &,
      unsigned int type = OEMCMolType::OEDefault)
```

Default and copy constructors.

type
The type of the molecule from the *OEMCMolType* namespace.

ClearBase

```
void ClearBase()
```

ClearMCMol

```
void ClearMCMol()
```

ClearMolBase

```
void ClearMolBase()
```

MCMol

```
const OEMCMolBase& MCMol() const
OEMCMolBase& MCMol()
```

SCMol

```
const OEMolBase& SCMol() const
OEMolBase& SCMol()
```

OEMolBase

```
class OEMolBase : public OESystem::OEBase
```

This class represents *OEMolBase*.

The *OEMolBase* class is the abstract interface for representing molecules within *OEChem*.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEConfBase*
- *OEMCMolBase*
- *OEQMolBase*

operator bool

```
operator bool() const
```

Determines whether the *OEMolBase* object contains any atoms. This method is equivalent to *OEMolBase.NumAtoms()* != 0.

Clear

```
void Clear()=0
```

Resets a molecule to its initial state. This method deletes all atoms (*OEAtomBase*) and bonds (*OEBondBase*) that are part of the molecule. Following an *OEMolBase.Clear*, the atom and bond indices assigned to new atoms and bonds may not be unique with those assigned prior to the *OEMolBase.Clear*.

ClearBase

```
void ClearBase()
```

Clear the generic data from the *OEBase* base class of this object. Equivalent to just calling *OEBase.Clear* without actually clearing away molecule data like atoms and bonds.

ClearCoords

```
void ClearCoords() =0
```

Clears the coordinates, resets stereo perception flags and sets the coordinate dimension to 0

See also

- *OEMolBase.GetCoords*
- *OEMolBase.SetCoords*

Compress

```
bool Compress()
```

See also

- Example program *RandomSample.cs*

Count

```
unsigned int Count(unsigned int) const = 0
```

CreateCopy

```
OESystem::OEBase* CreateCopy() const
```

DeleteAtom

```
bool DeleteAtom(OEAtomBase* atom)=0
```

Deletes an atom from a molecule. All bonds connected to the specified atom are also automatically deleted. Following an atom deletion, the specified *OEAtomBase* pointer, and the *OEBondBase* pointers of any incident bonds, are no longer valid and should not be used. The atom index associated with the atom, and the bond indices of any incident bonds, are no longer recognized by the *OEMolBase*, but the indices of all other atoms and bonds remain stable, and are unaffected by this call.

Note

After deleting atoms or bonds, it may be necessary to call the *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags* functions to update the 'in ring' and 'aromatic' properties of the atoms and bonds in the modified molecule.

DeleteBond

```
bool DeleteBond(OEBondBase* bond)=0
```

Deletes a bond from a molecule. Following a bond deletion, the specified *OEBondBase* pointer is no longer valid and should not be used. The bond index associated with the bond is no longer recognized by the *OEMolBase*, but the indices of all other bonds remain stable, and are unaffected by this call.

Note

After deleting atoms or bonds, it may be necessary to call the *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags* functions to update the 'in ring' and 'aromatic' properties of the atoms and bonds in the modified molecule.

DeleteGroup

```
bool DeleteGroup(OEGroupBase* group)
```

Deletes a group from a molecule. Following a group deletion, the specified *OEGroupBase* pointer is no longer valid and should not be used.

GetAtom

```
OEAtomBase* GetAtom(const OESystem::OEUnaryPredicate<OEChem::OEAtomBase> &) const = 0
```

Retrieves the first atom of a molecule that matches the specified atom predicate (*OEUnaryPredicate*). If the molecule does not contain an atom that matches, a NULL pointer, (*OEAtomBase**)0, is returned.

GetAtoms

```
OESystem::OEIterBase<OEAtomBase>* GetAtoms() const = 0
```

Returns an iterator over all the atoms of a molecule. By default, this returns all the atoms of the molecule in the order they were created. The ordering of the atoms returned by *OEMolBase.GetAtoms* may be modified by *OEMolBase.OrderAtoms*.

```
OESystem::OEIterBase<OEAtomBase>* GetAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase> &) const = 0
```

Returns an iterator over all of the atoms of a molecule that match the specified atom predicate (*OEUnaryPredicate*).

GetBond

```
OEBondBase* GetBond(const OEAtomBase* src, const OEAtomBase* dst) const
```

Retrieves the bond of a molecule between two specified atoms. If the two atoms, 'src' and 'dst' are not bonded together, then a NULL pointer, (*OEBondBase**)0, is returned

```
OEBondBase* GetBond(const OESystem::OEUnaryPredicate<OEChem::OEBondBase> &) const = 0
```

Retrieves the first bond of a molecule that matches the specified bond predicate. If the molecule does not contain a bond that matches the predicate, a NULL pointer, (*OEBondBase**)0, is returned.

GetBonds

```
OESystem::OEIterBase<OEBondBase>* GetBonds() const = 0
```

Returns an iterator over all the bonds of a molecule. By default, this returns all the bonds of the molecule in the order they were created. The ordering of the bonds returned by *OEMolBase.GetBonds* may be modified by *OEMolBase.OrderBonds*.

```
OESystem::OEIterBase<OEBondBase>* GetBonds(const OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase> &) const = 0
```

Returns an iterator over all of the bonds of a molecule that match the specified bond predicate (*OEUnaryPredicate*).

GetCoords

```
bool GetCoords(OEPlatform::OEHalfFloat *) const = 0
bool GetCoords(float *) const = 0
bool GetCoords(double *) const = 0
bool GetCoords(long double *) const = 0
```

Fills the specified array of floating point values with the Cartesian coordinates of all of the atoms in the molecule. The array argument must point to an array of at least 3* *OEMolBase.GetMaxAtomIdx()* elements. The X, Y and Z coordinates for the atom with index 'i' will be placed at offsets 3*i, 3*i+1 and 3*i+2 respectively.

```
bool GetCoords(const OEAtomBase *, OEPlatform::OEHalfFloat *) const = 0
bool GetCoords(const OEAtomBase *, float *) const = 0
bool GetCoords(const OEAtomBase *, double *) const = 0
bool GetCoords(const OEAtomBase *, long double *) const = 0
```

Fills the specified array of floating point values with the Cartesian coordinates of the specified atom. This array must be large enough to hold at least three values, corresponding to the X, Y and Z values for the given atom.

GetDimension

```
unsigned int GetDimension() const = 0
```

Returns the 'dimensionality' property of a molecule. The default value is zero, for unknown or no coordinates, 2 for 2-dimensional coordinates (such as depictions) and 3 for 3-dimensional coordinates. The 'dimensionality' property of a molecule may be set using the *OEMolBase.SetDimension* method.

This property is typically set by the appropriate molecular file format reader (0 for SMILES, 3 for MOL2 and 2 or 3 for MDL SD files *etc.*. . .) or by calling the *OESetDimensionFromCoords* function.

GetEnergy

```
double GetEnergy() const = 0
```

Returns the 'energy' property of a molecule. The default value is 0.0. The 'energy' property of a molecule may be set using the *OEMolBase.SetEnergy* method. Higher values indicate higher energies and therefore less-favorable or more-strained structures.

GetGroup

```
OEGroupBase* GetGroup(const OESystem::OEUnaryPredicate<OEChem::OEGroupBase>& pred) const
```

Retrieves the first group of a molecule that matches the specified group predicate (*OEUnaryPredicate*). If the molecule does not contain a group that matches, a NULL pointer, (*OEGroupBase**)0, is returned.

See also

- *OEHasAtomInGroup* predicate
- *OEHasBondInGroup* predicate
- *OEHasGroupType* predicate
- *OEIsMDLStereoGroup* predicate

GetGroups

```
OESystem::OEIterBase<OEGroupBase>* GetGroups() const
```

Returns an iterator over all the groups of a molecule.

```
OESystem::OEIterBase<OEGroupBase>* GetGroups(const OESystem::OEUnaryPredicate
↪<OEChem::OEGroupBase>& pred) const
```

Returns an iterator over all of the groups of a molecule that match the specified group predicate (*OEUnaryPredicate*).

See also

- *OEHasAtomInGroup* predicate
- *OEHasBondInGroup* predicate
- *OEHasGroupType* predicate
- *OEIsMDLStereoGroup* predicate

GetMaxAtomIdx

```
unsigned int GetMaxAtomIdx() const = 0
```

For performance reasons, atom indices are not guaranteed to be contiguous. *OEMolBase.GetMaxAtomIdx* returns a lower bound for the maximum atom index i.e. all indices returned by *OEAtomBase.GetIdx* of the molecule will be less than this value. The value returned by *OEMolBase.GetMaxAtomIdx* is always at least as large as *OEMolBase.NumAtoms*.

This method is useful for allocating temporary external data structures that need to be indexed by atom index, as returned by *OEAtomBase.GetIdx*.

GetMaxBondIdx

```
unsigned int GetMaxBondIdx() const = 0
```

For performance reasons, bond indices are not guaranteed to be contiguous. *OEMolBase.GetMaxBondIdx* returns a lower bound for the maximum bond index i.e. all indices returned by *OEBondBase.GetIdx* of the molecule will be less than this value. The value returned by *OEMolBase.GetMaxBondIdx* is always at least as large as *OEMolBase.NumBonds*.

This method is useful for allocating temporary external data structures that need to be indexed by bond index, as returned by *OEBondBase.GetIdx*.

GetTitle

```
const char* GetTitle() const
```

Returns the 'title' property of a molecule. The default value is the empty string. The title of a molecule may be set using the *OEMolBase.SetTitle* method.

See also

- Example programs *GetTitles.cs* and *MolExtract.cs*

HasPerceived

```
bool HasPerceived(unsigned int prop) const = 0
```

Returns whether a property is perceived for the given molecule.

property
This value has to be from the *OEPerceived* namespace.

See also

- *OEMolBase.ResetPerceived* method
- *OEMolBase.SetPerceived* method

IsDataType

```
bool IsDataType(const void *) const
```

IsDeleted

```
bool IsDeleted(OEBondBase *) const = 0
bool IsDeleted(OEAtomBase *) const = 0
```

IsRxn

```
bool IsRxn() const
```

Determines whether the molecule represents a reaction/transform. A `true` returned value indicates that the molecule represents a reaction, while `false` indicates the molecule is a simple connection table. The default value is `false`. The reaction property of a molecule may be set using the *OEMolBase.SetRxn* method.

NewAtom

```
OEAtomBase* NewAtom(unsigned int)=0
```

Creates a new atom in the molecule by specifying the element number from the *OEElemNo* namespace. This method does not create any bonds, and the returned created atom is always disconnected.

```
OEAtomBase* NewAtom(const OEAtomBase &)=0
```

Creates a new atom in the molecule and copies the atomic properties of the specified atom (atomic number, formal charge, implicit hydrogen count, *etc.*...) This method does not create any bonds, and the returned created atom is always disconnected.

NewBond

```
OEBondBase* NewBond(OEAtomBase* src, OEAtomBase* dst,
                    unsigned int order = 0)=0
```

Creates a new bond in the molecule. The 'src', 'dst' and 'order' arguments are used to specify the begin atom, the end atom and the bond order of the new bond. This method returns a pointer to the newly created *OEBondBase* object. The atoms specified as 'src' and 'dst' atoms must belong to the current molecule. Additionally, a NULL pointer, (*OEAtomBase**)0, may be passed as either the begin or end atom (but not both), allowing that atom to be specified later using *OEBondBase.SetBgn* or *OEBondBase.SetEnd* respectively. This helps when creating molecules for applications in which the ordering of atoms and/or bonds is significant.

Note

After creating new bonds, it may be necessary to call the *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags* functions to update the 'in ring' and 'aromatic' properties of the atoms and bonds in the modified molecule.

NewGroup

```
OEGroupBase *NewGroup(unsigned int type,
                      const std::vector<OEAtomBase *> &atoms)
OEGroupBase *NewGroup(unsigned int type,
                      const std::vector<OEBondBase *> &bonds)
OEGroupBase *NewGroup(unsigned int type,
                      const std::vector<OEAtomBase *> &atoms,
                      const std::vector<OEBondBase *> &bonds)
OEGroupBase* NewGroup(unsigned int type, const OEAtomBondSet &abset)
```

Creates a new group in the molecule with the given atoms or/and bonds.

type
The type associated with the created *OEGroupBase* object.

atoms
The pointer of the *OEAtomBase* objects that the group will hold.

bonds
The pointer of the *OEBondBase* objects that the group will hold.

abset
The *OEAtomBondSet* container that holds atom and bond pointers.

Note

An empty group can not be created. If there is no valid atom or/and bond pointers given, then the *OEMolBase.NewGroup* will return a NULL pointer.

See also

- *OEGroupType* namespace

NumAtoms

```
unsigned int NumAtoms() const = 0
```

Returns the number of atoms in the molecule.

NumBonds

```
unsigned int NumBonds() const = 0
```

Returns the number of bonds in the molecule.

NumGroups

```
unsigned int NumGroups() const
```

Returns the number of groups in the molecule.

OrderAtoms

```
bool OrderAtoms(const std::vector<OEAtomBase *> &)=0
```

Reorders the atoms of the molecule. This method modifies the order in which the atoms are visited by the iterator returned by *OEMolBase.GetAtoms*. This method does not affect the atom indices of any of the atoms of the molecule. This method does not affect the ordering of any other iterator, including bonds over a molecule (*OEMolBase.GetBonds*) , bonds over an atom (*OEAtomBase.GetBonds*) or neighboring atoms over an atom (*OEAtomBase.GetAtoms*).

OrderBonds

```
bool OrderBonds(const std::vector<OEBondBase *> &)=0
```

Reorders the bonds of a molecule. This method modifies the order in which the bonds are visited by the iterator returned by *OEMolBase.GetBonds*. This method does not affect the bond indices of any of the bonds of the molecule. This method does not affect the ordering of any other iterator, including atoms over a molecule (*OEMolBase.GetAtoms*), bonds over an atom (*OEAtomBase.GetBond*) or neighboring atoms over atom (*OEAtomBase.GetAtoms*).

ResetPerceived

```
void ResetPerceived() = 0
```

Invalidates all previously perceived properties specified in the *OEPerceived* namespace.

See also

- *OEMolBase.HasPerceived* method
- *OEMolBase.SetPerceived* method

SetCoords

```
bool SetCoords(const OEPlatform::OEHalfFloat* coords)=0
bool SetCoords(const float* coords)=0
bool SetCoords(const double* coords)=0
bool SetCoords(const long double* coords)=0
```

Sets the Cartesian coordinates for all of the atoms in a molecule. The array argument must point to an array of at least 3* *OEMolBase.GetMaxAtomIdx()* elements. The X, Y and Z coordinates for the atom with index 'i', should be placed at offsets `3*i`, `3*i+1` and `3*i+2` respectively.

See also

- *OEMolBase.ClearCoords*

```
bool SetCoords(const OEAtomBase *, const OEPlatform::OEHalfFloat *)=0
bool SetCoords(const OEAtomBase *, const float *)=0
bool SetCoords(const OEAtomBase *, const double *)=0
bool SetCoords(const OEAtomBase *, const long double *)=0
```

Sets the Cartesian coordinates of the specified atom to the values specified by the floating point array. The array argument must point to array of at least three values, representing the X, Y and Z Cartesian coordinates of the atom, respectively.

SetDimension

```
bool SetDimension(unsigned int)=0
```

Sets the 'dimensionality' property of a molecule. The default value is zero. The 'dimensionality' property of a molecule may be retrieved using the *OEMolBase.GetDimension* method. This property is typically set by the appropriate molecular file format reader, or by calling the *OESetDimensionFromCoords* function.

The 'dimensionality' property represents the dimensionality of the coordinates. This has the default value zero, for unknown or no coordinates, 2 for 2-dimensional coordinates (such as depictions) and 3 for 3-dimensional coordinates. This property is typically set by the file format reader indicating the dimensionality of the input file (`0` for SMILES, `3` for MOL2 and `2` or `3` for MDL SD files *etc.*. . .)

SetEnergy

```
bool SetEnergy(float)=0
bool SetEnergy(double)=0
```

Sets the 'energy' property of a molecule. The default value is `0.0`. The energy of a molecule may be retrieved using the *OEMolBase.GetEnergy* method.

SetPerceived

```
bool SetPerceived(unsigned int property, bool perceived) = 0
```

Sets whether a property is considered to be perceived for the given molecule.

property
: This value has to be from the *OEPerceived* namespace.

perceived
A boolean value that determines whether the given property is perceived.

Warning

The perception flags are used for performance in order to avoid perceiving properties that have already been calculated. It should be set to be 'true' only by the function that calculates the given property.

See also

- *OEMolBase.HasPerceived* method
- *OEMolBase.ResetPerceived* method

SetRxn

```
bool SetRxn(bool)
```

Sets the 'reaction' property of a molecule. The 'reaction' property of a molecule may be retrieved using the *OEMolBase.IsRxn* method.

SetTitle

```
bool SetTitle(const char *)=0
bool SetTitle(const std::string &)=0
```

Sets the 'title' property of a molecule. The default value is the empty string. The 'title' property of a molecule may be retrieved using the *OEMolBase.GetTitle* method.

See also

- Example program *SDFRename.cs*

Sweep

```
bool Sweep()=0
```

Renumbers the atom and bond indices sequentially. This method invalidates the atom and bond indices of all atoms in a molecule.

Note

This method doesn't guarantee that all atom and bond indices are sequential upon completion (some molecule implementations may treat *OEMolBase.Sweep* as a no-op).

UnCompress

```
bool UnCompress()
```

See also

- Example program *RandomSample.cs*

OEMolDatabase

```
class OEMolDatabase : public OESystem::OEBase
```

This class provides an abstraction for fast read-only random access to any molecular file format **OEChem** supports reading. The *Molecular Database Handling* chapter provides a description of the fundamentals of using this class.

When *opened* or *saved* to an `.oeb` file, the generic data on the object itself will be preserved through the *OEHeader* record in the file. The following methods can be used to access the generic since they are inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEMolDatabase()
OEMolDatabase(OEChem::oemolistream& ifs)
OEMolDatabase(const std::string& filename)
OEMolDatabase(OEPlatform::oeistream& istr, unsigned int format)
OEMolDatabase(const std::string& filename, unsigned int format)
OEMolDatabase(const std::string& filename, unsigned int format,
              unsigned int flavor)
OEMolDatabase(OEPlatform::oeistream& istr, unsigned int format,
              unsigned int flavor)
```

The default constructor will initialize the object to an empty state where most of the accessors methods will fail. It is expected that the user will later call *OEMolDatabase.Open* on the object.

The other constructors are convenience methods for opening the database on the given `filename` or stream with the specified `format` and `flavor`.

The database requires file storage to store the actual molecule data. Therefore, if given a *oeistream*, the stream's data will be copied to a temporary file. The temporary file will be automatically deleted by the destructor of this class.

oemolistream may not need a temporary file if they were opened on an actual file.

Clear

```
void Clear()
```

Removes all generic by calling through to *OEBase.Clear* as well as resetting the database back to an un-initialized state, the same state as the default constructor.

CreateCopy

```
OESystem::OEBase* CreateCopy() const
```

Creates a copy of the database pointing to the same backing file store as the original. The copy is invalid if the original is destroyed.

> **Warning**
>
> It is not recommended to use this method. It is only implemented to adhere to the *OEBase* API.

Delete

```
bool Delete(unsigned int idx)
bool Delete(const std::vector<unsigned int>& indices)
bool Delete(const OESystem::OEUnaryPredicate<OEMolBase>& pred)
bool Delete(const OESystem::OEUnaryPredicate<OEMCMolBase>& pred)
```

Mark the given molecules as "deleted" in the database in constant time, `O(1)`. Deleted molecules will not be written during a *Save* operation. Deleted molecules will return `false` when accessed from the *OEMolDatabase.GetMolecule* method.

GetDataType

```
const void* GetDataType() const
```

Return an opaque value representing the data type of this class.

GetFlavor

```
unsigned int GetFlavor() const
```

Returns `0` if uninitialized. Returns the flavor used to interpret the molecules in this database, pulled from the *OEIFlavor* namespace associated with the format returned from the *OEMolDatabase.GetFormat* method.

GetFormat

```
unsigned int GetFormat() const
```

Returns the format used to interpret the molecules in the file this database was opened on. The value returned is a member of the *the OEFormat namespace*.

GetIdx

```
unsigned int GetIdx(unsigned int row)
```

Returns the index for the nth `row` in the database. `row` should be less than *OEMolDatabase.NumMols* and the value returned will be less than *OEMolDatabase.GetMaxMolIdx*. By default, row numbers are the same as their index. Row numbers can be changed by deleting or re-ordering the database. If row numbers were altered before this call, this call is `O(GetMaxMolIdx())`. After the first call to `GetIdx`, subsequent calls are `O(1)`, until another alteration is made to the database.

This is the reverse operation from *OEMolDatabase.GetRow*.

GetIdxs

```
OESystem::OEIterBase<unsigned int>* GetIdxs() const
bool GetIdxs(std::vector<unsigned int>& indices) const
```

Returns all the valid indices in the order that is currently considered the database order by this object.

GetMaxMolIdx

```
unsigned int GetMaxMolIdx() const
```

Returns one more than the maximum molecule index this database currently contains.

GetMolecule

```
bool GetMolecule(OEMolBase& mol, unsigned int idx) const
bool GetMolecule(OEMCMolBase& mol, unsigned int idx) const
bool GetMolecule(oemolostream& ostr, unsigned int idx) const
```

Read the molecule located at index `idx` into the molecule `mol`. If the file format is a multi-conformer `.oeb` file, only the first will be read if `mol` is an *OEMolBase*. If there is a need for handling multi-conformer molecules, the *OEMCMolBase* overload should be used instead.

Warning

The overload that takes an *oemolostream* is deprecated as of 2015.Jun and will be removed in a future version of **OEChem TK**, please use *OEMolDatabase.WriteMolecule* instead.

The overload that takes an *oemolostream* will write bytes for the molecule record at index `idx` to the stream `ostr`. No attempt will be made to interpret the data as a molecule and verify whether it is truly a parsable molecule. The molecule will be in whatever format the database was opened on.

GetMoleculeString

```
std::string GetMoleculeString(unsigned int idx) const
```

Return the chunk of data from the molecule file that corresponds to the molecule record at index `idx`.

GetRow

```
unsigned int GetRow(unsigned int idx)
```

Return where the database considers the molecule record `idx` to reside in the global order of the database. `idx` should be less than *OEMolDatabase.GetMaxMolIdx* and the value returned will be less than *OEMolDatabase.NumMols*. By default, row numbers are the same as their index. Row numbers can be changed by deleting or re-ordering the database. If row numbers were altered before this call, this call is `O(GetMaxMolIdx())`. After the first call to `GetIdx`, subsequent calls are `O(1)`, until another alteration is made to the database.

This is the reverse operation from *OEMolDatabase.GetIdx*.

GetTitle

```
std::string GetTitle(unsigned int idx) const
```

Returns the title for the molecule at index `idx` in the database. This is the same value that is returned by *OEMolBase.GetTitle*, but without the need to parse the whole molecule record into an *OEMolBase*.

GetTitles

```
OESystem::OEIterBase<const std::string>* GetTitles() const
```

Returns all the titles for the database as would be returned by *OEMolDatabase.GetTitle* in database index order.

IsDataType

```
bool IsDataType(const void *) const
```

Returns whether the opaque data passed represents the same data type as this class.

IsDeleted

```
bool IsDeleted(unsigned int idx) const
```

Returns whether the database index `idx` has already been deleted by a previous call to *OEMolDatabase.Delete*.

NumMols

```
unsigned int NumMols() const
```

Returns the number of molecule **records** in the database.

Note

The number of valid molecules in the database might be less than the number returned by *OEMolDatabase.NumMols*. The reason is that the molecule file is not fully parsed when invoking the *OEMolDatabase.NumMols* function.

Open

```
bool Open(OEChem::oemolistream& ifs)
bool Open(const std::string& filename)
bool Open(const std::string& filename, unsigned int format)
bool Open(OEPlatform::oeistream& istr, unsigned int format)
bool Open(const std::string& filename, unsigned int format,
          unsigned int flavor)
bool Open(OEPlatform::oeistream& istr, unsigned int format,
          unsigned int flavor)
```

Initializes the database to the file name given. The database requires file storage to store the actual molecule data. Therefore, if given a *oeistream*, the stream's data will be copied to a temporary file. The temporary file will be automatically deleted by the destructor of this class.

oemolistream may not need a temporary file if they were opened on an actual file.

This method is expensive, it may require scanning the whole file to determine the proper file offsets to the individual molecule records. To speed up the `Open` operation, `.idx` files can be generated by *OECreateMolDatabaseIdx* or *OEMolDatabase.Save*. If `Open` detects the presence of an `.idx` file parallel to the database file, the file offsets will be read directly from this file instead, possibly saving the need to stream the entire database from disk.

See also

Index Files section

Order

```
bool Order(const std::vector<unsigned int>& indices)
```

Re-orders the molecules in the database according to the order of the indices specified by `indices`. Subsequent calls to *OEMolDatabase.GetIdx* or *OEMolDatabase.GetRow* will return this order. Subsequent calls to *OEMolDatabase.Save* will output the database in this order.

Save

```
bool Save(const std::string& filename, unsigned int format) const
bool Save(OEPlatform::oeostream& ostr, unsigned int format) const
bool Save(const std::string& filename, unsigned int format,
          unsigned int flavor) const
bool Save(OEPlatform::oeostream& ostr, unsigned int format,
          unsigned int flavor) const
bool Save(OEChem::oemolostream& ofs,
          const OEMolDatabaseSaveOptions& opts = OEMolDatabaseSaveOptions()) const
```

(continues on next page)

(continued from previous page)

```
bool Save(const std::string& filename,
          const OEMolDatabaseSaveOptions& opts = OEMolDatabaseSaveOptions()) const
```

Save the database to the file specified by `filename` in the `format` and `flavor` specified. The data may be written directly to the stream `ostr` as well.

This method is heavily optimized for the case whenever the output file format is the same as the input file format used to initialize the database. If the same file format is used, the molecule record's bytes will be streamed directly, without an intermediate *OEMolBase* to do the conversion.

See also

- *OEMolDatabaseSaveOptions* class for more options when saving a database to a file.
- *OEGetMolDatabaseIdxFileName* to retrieve the default filename used for the index file generated.

Warning

Since *OEMolDatabase* depends on backing file storage. Therefore, if the contents of the file change after *OEMolDatabase.Open* completes, the behavior is undefined. It is very likely the file data itself will be corrupted. For this reason, *OEMolDatabase.Save* will fail to write to the same file name used to open the database object. This protects against the common case, but does not protect against the general case of file changing during access.

WriteMolecule

```
bool WriteMolecule(oemolostream& ostr, unsigned int idx) const
```

Write bytes for the molecule record at index `idx` to the stream `ostr`. No attempt will be made to interpret the data as a molecule and verify whether it is truly a parsable molecule. The molecule will be in whatever format the database was opened on. Returns `false` if the *oemolostream* is not the exact same file format as the *OEMolDatabase* or any input/output operation fails.

OEMolDatabaseSaveOptions

```
class OEMolDatabaseSaveOptions
```

The class used to contain additional options for the *OEMolDatabase.Save* method.

Constructors

```
OEMolDatabaseSaveOptions()
OEMolDatabaseSaveOptions(unsigned int format, unsigned int flavor,
                         bool writeIdx,
                         const std::string& idxFileName)
```

The default constructor initializes the object to the following:

format *OEFormat.UNDEFINED*

flavor `0`

writeIdx `true`

idxFileName Empty string

GetFlavor

```
unsigned int GetFlavor() const
```

Returns a constant from the *OEOFlavor* namespace for the format being saved to.

GetFormat

```
unsigned int GetFormat() const
```

Returns a constant from the *the OEFormat namespace* namespace.

GetIdxFileName

```
const std::string& GetIdxFileName() const
```

Returns the name of the index file being written to. If this file name is empty and *OEMolDatabaseSaveOptions.GetWriteIdx* returns `true`, the file name will be automatically determined with the same rules as *OEGetMolDatabaseIdxFileName*.

See also

- *OEGetMolDatabaseIdxFileName*
- *OECreateMolDatabaseIdx*

GetWriteIdx

```
bool GetWriteIdx() const
```

Returns whether an index file should be created as well.

SetFlavor

```
void SetFlavor(unsigned int flvr)
```

Specifies what the file format flavor the database should be saved to. `flvr` should be a value from the *OEOFlavor* namespace.

SetFormat

```
void SetFormat(unsigned int fmt)
```

Specifies what file format the database should be saved to. `fmt` should be a value from the *the OEFormat namespace*.

SetIdxFileName

```
void SetIdxFileName(const std::string& f)
```

Specifies what the index file name should be.

See also

Index Files

SetWriteIdx

```
void SetWriteIdx(bool w)
```

Specifies whether an index file should be created during the database save operation.

OEMolDBException

```
class OEMolDBException : public std::exception
```

The exception that is raised when the *OEMolDatabase constructors* fail.

Constructors

```
OEMolDBException(const std::string& msg)
```

Construct the exception with the `msg`.

what

```
const char* what() const
```

Returns the message used to construct this exception.

OEMolGDPropertyFxn

```
class OEMolGDPropertyFxn : public OEMolPropertyFxn
```

This class can be used to access molecule properties defined as generic data in a molecule.

The following methods are publicly inherited from *OEMolPropertyFxn*:

CreateCopy *Get* *Has*

Constructors

```
OEMolGDPropertyFxn(const std::string& tag)
OEMolGDPropertyFxn(const OEMolGDPropertyFxn &)
```

Default and copy constructors. The tag parameter referes to the generic data tag corresponding to the property of interest.

oemolistream

```
class oemolistream : public oemolstreambase
```

The *oemolistream* class provides a stream-like abstraction for reading molecules from files, strings or standard input. The *oemolistream* maintains the format and flavor of molecular reading for the stream. It also manages the conversion between multi-conformer molecules and single conformer molecules in cases where the molecule read into is not compatible with the file format (in the sense of a multi-conformer file format being read into a single-conformer molecule, or a single-conformer file format being read into a multi-conformer molecule). The *oemolistream* is capable of uncompressing gzip files while reading.

The following methods are publicly inherited from *oemolstreambase*:

ClearCSVHeader	*GetFlavor*	*SetFormat*
GetBinaryIOHandler	*GetFormat*	*close*
GetCSVHeader	*Getgz*	*open*
GetFileName	*SetFlavor*	

Constructors

```
oemolistream()
```

Default constructor. Creates a new *oemolistream* object that is connected to the standard input.

```
explicit oemolistream(const char* fname)
explicit oemolistream(const std::string& fname)
```

Creates a new *oemolistream* object and opens the file specified by the given name (‘fname’).

```
explicit oemolistream(OEPlatform::oeistream *, bool owned = true)
```

Creates a new *oemolistream* object from an existing *oeistream* object. The second optional argument is used to indicate whether the new *oemolistream* now owns the given *oeistream* and is therefore responsible for closing and destroying it when it itself is closed and/or destroyed.

Warning

Only in C++ can an *oemolistream* object own an existing *oeistream* object.

See also

To associate a file or a stream with an *oemolistream* after it has been created, see the *oemolstreambase.open* method.

operator bool

```
operator bool() const
```

GetConfTest

```
const OEConfTestBase & GetConfTest() const
```

GetOEGraphMols

```
GetOEGraphMols() -> <generator of OEGraphMol objects>
```

Returns a generator over all the molecules in the input stream in the current order. Each molecule will be an *OEGraphMol* object. If opened on a multi-conformer `.oeb` file, only the first conformer will be returned for each molecule record.

> **Note**
>
> Unlike `OEChem::OEMolDatabase::GetOEGraphMols`, a reference to each molecule will be used for each iteration to avoid unnecessary copying of molecules.

GetMCMolBases

```
OESystem::OEIterBase<OEChem::OEMCMolBase>* GetMCMolBases(bool single = false)
```

GetMolBases

```
OESystem::OEIterBase<OEChem::OEMolBase>* GetMolBases()
```

GetReadFailureCt

```
unsigned GetReadFailureCt() const
```

IncrementReadFailureCt

```
void IncrementReadFailureCt()
```

ResetReadFailureCt

```
void ResetReadFailureCt()
```

SetConfTest

```
bool SetConfTest(const OEConfTestBase &)
```

Sets the functor class which is used to compare incoming graphs to determine whether they should be placed as conformers of a multi-conformer molecule or be returned individually as single molecules. The default conformer test never places separate graphs into a multi-conformer molecule (*OEDefaultConfTest*).

See also

- *OEAbsCanonicalConfTest*
- *OEAbsoluteConfTest*
- *OEConfTestBase*
- *OEDefaultConfTest*
- *OEIsomericConfTest*

SetFlavor

```
bool SetFlavor(unsigned int format, unsigned int flavor)
```

Sets the file flavor for a given format associated with the *oemolistream* object. The set of valid formats are defined in the *the OEFormat namespace*. The set of valid bitmasks flavors are defined in the *OEIFlavor* namespace. The current flavor can be queried using the *oemolstreambase.GetFlavor* method. Each format has its own specific flavor which must be set separately. The *oemolistream* constructors call the *OESetDefaultFlavors* function to set the flavors for all of the formats to their default state.

SetFormat

```
bool SetFormat(unsigned int format)
```

Sets the file format associated with the *oemolistream* object. The set of valid formats are defined in the *the OEFormat namespace*. By default, when reading from standard input, the associated file format is *OEFormat.SMI*. The file format property of an *oemolistream* may be retrieved using the *oemolstreambase.GetFormat* method.

Note

The file format property is also set automatically by *oemolstreambase.open* based upon the file extension of the specified filename.

SetTracer

```
bool SetTracer(OESystem::OETracerBase* tracer, bool owntracer)
```

Registers the tracer, `tracer`, to report the current progress of the *oemolistream* object on reading molecules. The second optional argument is used to indicate whether the *oemolistream* now owns the given *OETracerBase* object and is therefore responsible for closing and destroying it when it itself is closed and/or destroyed.

Warning

Only in C++ can an *oemolistream* object own an existing *OETracerBase* object.

See also

- *OETracerBase*

Setgz

```
bool Setgz(bool gz, OEPlatform::oeistream* sptr = 0,
           bool owned = false)
```

Specifies that the contents of the *oemolistream* object are to be treated as compressed by GNU gzip compressed. The decompression takes place on-the-fly. Usually the 'gz' property of a *oemolistream* object is determined implicitly from the file extension used to open the stream for reading. The current 'gz' property of the *oemolistream* object can be retrieved using the *oemolstreambase.Getgz* method.

close

```
void close()
```

Closes the *oemolistream* object. This method may be safely called multiple times. However *oemolistream.close* method is called from within the *oemolistream* destructor and therefore it is not necessary to call this explicitly under most circumstances.

eof

```
bool eof() const
```

getbyte

```
int getbyte()
```

getline

```
bool getline(char* buffer, oesize_t max)
```

OpenByteArray

```
boolean OpenByteArray(byte[] buffer)
boolean OpenByteArray(byte[] buffer, uint len)
```

Allows the *oemolistream* to read from a buffer in memory, instead of from a file or standard input.

Internally, the *oemolistream.OpenByteArray* method make a copy of the specified file contents, allowing the *oemolistream* to continue to function independently of whether the original buffer is later modified or deallocated.

> **Note**
>
> Prior to calling *oemolistream.OpenByteArray* it is necessary to specify the format and whether or not the format is compressed with gzip. These can be set with the *oemolistream.SetFormat* and *oemolistream.Setgz* methods respectively.

peekbyte

```
int peekbyte()
```

read

```
bool read(unsigned char* buffer, oesize_t len)
```

rewind

```
void rewind()
```

seek

```
void seek(oefpos_t pos)
```

Moves the position of the next valid read to the position indicated. This function takes account of gzip streams and molecule caching.

size

```
oefpos_t size()
```

Returns the size of the input stream if applicable to the current stream. The return type is a portable file-system pointer type.

skip

```
bool skip(oefpos_t len)
```

skipbyte

```
bool skipbyte()
```

skipline

```
bool skipline()
```

tell

```
oefpos_t tell()
```

Returns the current position of the next read. This function accounts for molecular caching.

> **Note**
>
> If you are reading an `.oeb` file that was written as multi-conformer molecules and is being read with single conformer molecules, all of the conformers are read into cache at once, and the pointer will point to the beginning of a multi-conformer molecule rather than to a conformer inside a molecule.

oemolithread

```
class oemolithread : public oemolthreadbase
```

The following methods are publicly inherited from *oemolthreadbase*:

operator bool	*GetMol*	*PutMol*

The following methods are publicly inherited from *oemolstreambase*:

ClearCSVHeader	*GetFlavor*	*SetFormat*
GetBinaryIOHandler	*GetFormat*	*close*
GetCSVHeader	*Getgz*	*open*
GetFileName	*SetFlavor*	

Constructors

```
oemolithread()
explicit oemolithread(const char* fn)
explicit oemolithread(const std::string& fn)
```

Default and copy constructors.

GetIStream

```
OEPlatform::oeistream* GetIStream()
```

GetMolStream

```
oemolistream* GetMolStream()
```

SetFlavor

```
bool SetFlavor(unsigned int format, unsigned int flavor)
```

SetFormat

```
bool SetFormat(unsigned int format)
```

close

```
void close()
```

openstring

```
bool openstring(const std::string& str, bool gz = false)
bool openstring(const unsigned char* buffer, oesize_t len,
                bool gz = false)
```

oemolostream

```
class oemolostream : public oemolstreambase
```

The *oemolostream* class provides a stream-like abstraction for writing molecules to files, streams or standard output. The *oemolostream* maintains the format and flavor of the writer used when writing molecules. The *oemolostream* is capable of compressing gzip files while writing.

The following methods are publicly inherited from *oemolstreambase*:

ClearCSVHeader	*GetFlavor*	*SetFormat*
GetBinaryIOHandler	*GetFormat*	*close*
GetCSVHeader	*Getgz*	*open*
GetFileName	*SetFlavor*	

Constructors

```
oemolostream()
```

Creates a new *oemolostream* object that is connected to standard output.

```
explicit oemolostream(const char* fname)
explicit oemolostream(const std::string& fname)
```

Creates a new *oemolostream* object and opens the file specified by the given name ('fname').

```
explicit oemolostream(OEPlatform::oeostream *, bool owned = true)
```

Creates a new *oemolostream* object from the existing *oeostream* object. The second optional argument is used to indicate whether the new *oemolostream* now owns the given *oeostream* and is therefore responsible for closing and destroying it when it itself is closed and/or destroyed.

> **Warning**
>
> Only in C++ can an *oemolostream* object own an existing *oeostream* object.

See also

To associate a file or a stream with an *oemolostream* after it has been created, see the *oemolstreambase.open* method.

operator bool

```
operator bool() const
```

GetByteArray

```
byte[] GetByteArray()
```

Returns the contents of the in-memory buffer associated with the *oemolostream* object. The *oemolostream* object must previously have been opened for writing to an in-memory buffer using the *oemolostream.openstring* method.

SetFlavor

```
bool SetFlavor(unsigned int format, unsigned int flavor)
```

Sets the file flavor for a given format associated with the *oemolostream* object. The set of valid formats are defined int the *the OEFormat namespace*. The set of valid bitmasks flavors are defined in the *OEOFlavor* namespace. The current flavor can be queried using the *oemolstreambase.GetFlavor* method. Each format has its own specific flavor which must be set separately. The *oemolostream* constructors calls the *OESetDefaultFlavors* function to set the flavors for all of the formats to their default state.

SetFormat

```
bool SetFormat(unsigned int format)
```

Sets the file format associated with the *oemolostream* object. The set of valid formats are defined in the *the OEFormat namespace*. By default, when writing to standard output, the associated file format is *OEFormat.SMI*. The file format property of an *oemolistream* may be retrieved using the *oemolstreambase.GetFormat* method.

Note

The file format property is also set automatically by *oemolstreambase.open* based upon the file extension of the specified filename.

SetString

```
void SetString(const char* c)
void SetString(const unsigned char* c, oesize_t size)
```

Allows the *oemolostream* to be opened for writing to a buffer and specify the initial contents of the buffer.

The contents of in-memory buffer when writing to a string may be obtained using the *oemolostream.GetByteArray* method.

Setgz

```
bool Setgz(bool gz, OEPlatform::oeostream* sptr = 0, bool = false)
```

Specifies that the contents of the *oemolostream* object should be GNU gzip compressed The compression takes place on-the-fly. Usually the 'gz' property of a *oemolostream* object is determined implicitly from the file extension used to open the stream for writing. The current 'gz' property of the *oemolostream* object can be retrieved using the *oemolstreambase.Getgz* method.

close

```
void close()
```

Closes the *oemolostream* object. This method may be safely called multiple times.

flush

```
oemolostream& flush()
```

openstring

```
bool openstring()
bool openstring(bool gzip)
```

Allows the *oemolostream* to write to a buffer in memory, instead of to a file or standard output.

Calling *oemolostream.openstring* instructs the *oemolostream* object to start accumulating output in a buffer, which can later be retrieved by calling the *oemolostream.GetByteArray* method.

The form that accepts a 'gzip' argument can be used to instruct the *oemolostream* object to compress the contents of the in-memory buffer.

put

```
oemolostream& put(char c)
```

tell

```
oefpos_t tell()
```

write

```
oemolostream& write(const char* str, oesize_t size)
```

oemolothread

```
class oemolothread : public oemolthreadbase
```

The following methods are publicly inherited from *oemolthreadbase*:

operator bool	*GetMol*	*PutMol*

The following methods are publicly inherited from *oemolstreambase*:

ClearCSVHeader	*GetFlavor*	*SetFormat*
GetBinaryIOHandler	*GetFormat*	*close*
GetCSVHeader	*Getgz*	*open*
GetFileName	*SetFlavor*	

Constructors

```
oemolothread()
explicit oemolothread(const char* fn)
explicit oemolothread(const std::string& fn)
```

Default and copy constructors.

GetMolStream

```
oemolostream* GetMolStream()
```

GetOStream

```
OEPlatform::oeostream* GetOStream()
```

GetString

```
std::string GetString()
```

SetFlavor

```
bool SetFlavor(unsigned int format, unsigned int flavor)
```

SetFormat

```
bool SetFormat(unsigned int format)
```

close

```
void close()
```

openstring

```
bool openstring(bool gz = false)
```

OEMolParameter

```
class OEMolParameter : public OESystem::OEPointerParameter<OEMolBase>
```

The *OEMolParameter* represents parameter that has value of *OEMolBase* type.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

The *OEMolParameter* class defines the following public methods:

- *GetSetting*
- *GetValue*
- *SetValue*

Constructors

```
OEMolParameter()
OEMolParameter(const std::string& name)
OEMolParameter(const OEMolParameter&)
```

Default and copy constructors.

Constructs an *OEMolParameter* instance using the specified set of parameters.

GetSetting

```
const OEChem::OEMolBase& GetSetting() const
```

Returns the current setting of the parameter.

GetValue

```
const OEChem::OEMolBase& GetValue() const
```

Returns the current user-defined value of the parameter.

SetValue

```
bool SetValue(OEChem::oemolistream& fstream)
bool SetValue(const OEChem::OEMolBase& mol)
```

Sets the user-defined value of the parameter. For the overload with the `OEChem::oemolistream`, the first molecule is used as the value.

OEMolReaderParameter

```
class OEMolReaderParameter : public OESystem::OEStringParameter
```

The *OEMolReaderParameter* represents parameter that has `string` value representing a readable molecule file format.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*

- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEMolReaderParameter()
OEMolReaderParameter(const std::string& name,
                     const bool is3D = false)
OEMolReaderParameter(const OEMolReaderParameter&)
```

Default and copy constructors.

Constructs an *OEMolReaderParameter* instance using the specified set of parameters. The optional boolean argument referes to if the readable file format must contain 3D molecule.

OEMolWriterParameter

```
class OEMolWriterParameter : public OESystem::OEStringParameter
```

The *OEMolWriterParameter* represents parameter that has `string` value representing a writable molecule file format.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*

- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEMolWriterParameter()
OEMolWriterParameter(const std::string& name,
                     const bool is3D = false)
OEMolWriterParameter(const OEMolWriterParameter&)
```

Default and copy constructors.

Constructs an *OEMolWriterParameter* instance using the specified set of parameters. The optional boolean argument referes to if the writeable file format must be able to write 3D molecule.

OEMolPropertyFxn

```
class OEMolPropertyFxn
```

The *OEMolPropertyFxn* is an abstract base class. This class defines the interface for functions which can be used to access molecule properties defined as SDF file data fields or generic data in a molecule.

The following classes derive from this class:

- *OEMolGDPropertyFxn*
- *OEMolSDPropertyFxn*
- *OEMolTaggedPropertyFxn*

CreateCopy

```
OEMolPropertyFxn* CreateCopy() const = 0
```

A virtual constructor which allows copying of concrete derived objects using a reference to this base class.

Get

```
double Get(const OEChem::OEMolBase& mol) const = 0
double Get(const OEChem::OEMCMolBase& mol) const = 0
```

Returns the property value for the specified molecule. Method returns a large number (limiting large double in C++) when a value is not found.

Has

```
bool Has(const OEChem::OEMolBase& mol) const = 0
bool Has(const OEChem::OEMCMolBase& mol) const = 0
```

Checks if the property value exists for the specified molecule. Returns `True` if a value exists.

OEMolPropertyList

```
class OEMolPropertyList
```

This class is a container for multiple *OEMolPropertyFxn* objects.

Constructors

```
OEMolPropertyList()
OEMolPropertyList(const OEMolPropertyList &)
```

Default and copy constructors.

Add

```
void Add(const OEMolPropertyFxn &)
```

Adds a copy of the specified property function to the list.

Clear

```
void Clear()
```

Clears the list of property functions.

CreateCopy

```
OEMolPropertyList* CreateCopy() const
```

Returns a copy of this *OEMolPropertyList* object.

GetProperty

```
double GetProperty(const OEChem::OEMolBase &, size_t index) const
double GetProperty(const OEChem::OEMCMolBase &, size_t index) const
```

Returns the property value for the specified molecule, for the function located at the specified index in the list. Method returns a large number (limiting large double in C++) when a value is not found.

GetSize

```
size_t GetSize() const
```

Returns the current number of *OEMolPropertyFxn* objects in the list.

HasProperties

```
bool HasProperties(const OEChem::OEMolBase &) const
bool HasProperties(const OEChem::OEMCMolBase &) const
```

Checks if the all of the property values exist for the specified molecule. Returns `False` if at least of the property value does not exist.

Attention

This API is currently available in C++ and Python.

OEMolRecord

See also

- *OEMolRecord* chapter

OEMolSDPropertyFxn

```
class OEMolSDPropertyFxn : public OEMolPropertyFxn
```

This class can be used to access molecule properties defined as SD data in a molecule.

The following methods are publicly inherited from *OEMolPropertyFxn*:

CreateCopy	*Get*	*Has*

Constructors

```
OEMolSDPropertyFxn(const std::string& tag)
OEMolSDPropertyFxn(const OEMolSDPropertyFxn &)
```

Default and copy constructors. The tag parameter referes to the SD data tag corresponding to the property of interest.

oemolstreambase

```
class oemolstreambase
```

The *oemolstreambase* is the abstract base class for reading and writing molecules.

The following classes derive from this class:

- *oemolistream*
- *oemolithread*
- *oemolostream*
- *oemolothread*
- *oemolthreadbase*

ClearCSVHeader

```
void ClearCSVHeader()
```

Clears out the *OECSVHeader* object owned by this stream allowing an alternative set of column names to be initialized.

GetBinaryIOHandler

```
OESystem::OEBinaryIOHandlerBase& GetBinaryIOHandler() const
```

GetCSVHeader

```
OECSVHeader& GetCSVHeader()
```

Returns a reference to the *OECSVHeader* object owned by this stream. The first molecule read or written to this stream will initialize the *OECSVHeader* object to the set of SD data tags and the order in which to read or write them.

GetFileName

```
const std::string& GetFileName() const
```

Returns the file name used in the previous call to *oemolstreambase.open*. An empty string is returned if *oemolstreambase.open* has not been called yet or *oemolstreambase.close* has been called.

GetFlavor

```
unsigned int GetFlavor(unsigned int format)
```

Returns the file flavor associated with the format for the *oemolstreambase* object. The set of valid formats are defined in the *the OEFormat namespace*. The set of valid bitmasks flavors of input streams are defined in the *OEIFlavor* namespace. The set of valid bitmasks flavors of output streams are defined in the *OEOFlavor* namespace.

The flavor for any format can be set using the *oemolstreambase.SetFlavor* method. The default flavors are automatically set by the *oemolstreambase* constructors.

GetFormat

```
unsigned int GetFormat() const
```

Returns the file format associated with the output *oemolstreambase* object. The set of valid formats are defined in the *the OEFormat namespace*.

By default, when writing to standard output or when reading from standard input the associated file format is *OEFormat.SMI*.

The file format property can be set using the *oemolstreambase.SetFormat* method.

Note

The file format property is also set automatically by *oemolstreambase.open* based upon the file extension of the specified filename.

Getgz

```
bool Getgz()
```

Returns whether the stream is writing to or reading from a gzip compressed *oemolstreambase*.

SetFlavor

```
bool SetFlavor(unsigned int format, unsigned int flavor)=0
```

This pure virtual method is designed to set the flavor for a given format associated with an input/output molstream.

See also

- *oemolistream.SetFlavor*
- *oemolithread.SetFlavor*
- *oemolostream.SetFlavor*
- *oemolothread.SetFlavor*

SetFormat

```
bool SetFormat(unsigned int format)=0
```

This pure virtual method is designed to set the format associated with an input/output molstream.

See also

- *oemolistream.SetFormat*
- *oemolithread.SetFormat*
- *oemolostream.SetFormat*
- *oemolothread.SetFormat*

Note

Setting the molstream format to .gz or .oez when it was previously a different (or default) format re-initializes the OEB handlers in the molstream. This means, any explicit changes to the OEB handlers for the stream should be made after setting the format.

close

```
void close()
```

This method clears the *OECSVHeader* object owned by this stream, and resets the file name returned by *oemolstreambase.GetFileName* back to an empty string. It is expected that the user will not need to call this method directly, but rely on the following subclass methods to call this function:

- *oemolistream.close*
- *oemolithread.close*
- *oemolostream.close*
- *oemolothread.close*

open

```
bool open()
```

The input molstreams (*oemolistream*, *oemolithread*) read from to standard input. The output molstreams (*oemolostream*, *oemolothread*) write to standard input. The default file format for both input and output molstreams are *OEFormat.SMI*. Calling *oemolstreambase.open* (no arguments) on input or output molstreams will force the stream format to the default *OEFormat.SMI* and the gzip compression property to false.

```
bool open(const char* fname)
bool open(const std::string& fname)
```

In case of an **input molstream** (*oemolistream*, *oemolithread*), this method opens a file , specified by its filename ('fname') for reading. The file format property of the input molstream is set based upon the extension of the given filename. If the file extension isn't recognized, a warning is issued and the existing file format is unchanged. If the filename is appended with `.gz`, the input molstream will decompress the file on-the-fly when reading. If only a file extension is used as the filename (`.oeb.gz`), then the standard input is opened with the format specified by the given extensions.

In case of an **output molstream** (*oemolostream*, *oemolothread*), this method creates (or overwrite) a file , specified by its filename ('fname') for writing. The file format property of the output molstream is set based upon the extension of the given filename. If the filename is appended with `.gz`, the output molstream will compress the file on-the-fly when writing. If the file extension isn't recognized, a warning is issued and the existing file format is unchanged. If only a file extension is used as the filename (`.oeb.gz`), then the standard output is opened with the format specified by the given extensions.

In case of both **input** and **output molstream**, the filename based file format may be overridden by calling *oemolstreambase.SetFormat* explicitly with the desired file format.

OEMolStyleColorer

```
class OEMolStyleColorer
```

This class represents *OEMolStyleColorer*.

The *OEMolStyleColorer* class is the abstract interface for coloring a 3D molecule within *OEChem*. The *OEMolStyleColorer* is set within a *OE3DMolStyle* utilizing *OE3DMolStyle.SetAtomColorer*, *OE3DMolStyle.SetAtomLabelColorer*, *OE3DMolStyle.SetGridContourColorer*, *OE3DMolStyle.SetProteinColorer*, *OE3DMolStyle.SetProteinColorer*, and *OE3DMolStyle.SetSurfaceColorer*. The *OEMolStyleColorer* can be applied to the initial molecule, but can also applied to its conformers and atoms. The hierarchy between the different *OEMolStyleColorers* dictates that the settings specified on the atom supersede those set on the conformer, and likewise, the settings specified on the conformer will override those on the molecule.

Constructors

```
OEMolStyleColorer()
OEMolStyleColorer(unsigned int colorScheme)
OEMolStyleColorer(const char* propertyName)
OEMolStyleColorer(const OESystem::OEColor& col)
OEMolStyleColorer(const OEMolStyleColorer& other)
OEMolStyleColorer(const std::string& propertyName)
OEMolStyleColorer(unsigned int colorScheme,
                  const OESystem::OEColor& defaultColor)
OEMolStyleColorer(const char* propertyName,
                  const OESystem::OEColor& defaultColor)
OEMolStyleColorer(const std::string& propertyName,
                  const OESystem::OEColor& defaultColor)
```

Default and copy constructors.

AddColor

```
bool AddColor(int value, const OESystem::OEColor& col)
bool AddColor(double value, const OESystem::OEColor& col)
bool AddColor(unsigned int value, const OESystem::OEColor& col)
bool AddColor(const std::string& value,
              const OESystem::OEColor& col)
```

Adds OEColor to OEMolStyleColorer.

ClearColors

```
void ClearColors()
```

Clears all of the color maps in the OEMolStyleColorer.

GetColor

```
OESystem::OEColor GetColor() const
OESystem::OEColor GetColor(int val) const
OESystem::OEColor GetColor(double val) const
OESystem::OEColor GetColor(unsigned int val) const
OESystem::OEColor GetColor(const std::string& val) const
OESystem::OEColor GetColor(double val, double scaleMin,
                           double scaleMax) const
```

Returns a OEColor set in the OEMolStyleColorer. The function with no arguments will return the default color set in the OEMolStyleColorer. The functions with arguments will search for the color in the repected map based of the inputted value.

GetColorScheme

```
unsigned int GetColorScheme() const
```

Returns a integer that specifies the color scheme set in the OEMolStyleColorer.

GetDataTypes

```
unsigned int GetDataTypes() const
```

Returns a integer that specifies the data type set in the OEMolStyleColorer.

GetDefaultColor

```
OESystem::OEColor GetDefaultColor() const
```

Returns the OEColor set as the default color in the OEMolStyleColorer.

GetNumColors

```
unsigned int GetNumColors() const
```

Returns the number of colors set in all of the maps of the OEMolStyleColorer.

GetPropertyName

```
std::string GetPropertyName() const
```

Returns a string that specifies the color scheme name set in the OEMolStyleColorer.

GetValues

```
bool GetValues(std::vector<int>& vals) const
bool GetValues(std::vector<double>& vals) const
bool GetValues(std::vector<std::string>& vals) const
```

Returns a boolean indicating if the specified map has any data. The map that is searched is indicated by the data being stored in the vector. This data is pushed into the reference vector.

SetDefaultColor

```
void SetDefaultColor(const OESystem::OEColor& col)
```

Sets an OEColor as the default color in the OEMolStyleColorer.

See also

- *OE3DMolStyle*

OEMolTaggedPropertyFxn

```
class OEMolTaggedPropertyFxn : public OEMolPropertyFxn
```

This class can be used to access molecule properties defined as either SD data or generic data in a molecule. Methods of this class checks for SD data field first, and only looks for the generic data if a corresponding SD data field is not found.

The following methods are publicly inherited from *OEMolPropertyFxn*:

CreateCopy	*Get*	*Has*

Constructors

```
OEMolTaggedPropertyFxn(const std::string& tag)
OEMolTaggedPropertyFxn(const OEMolTaggedPropertyFxn &)
```

Default and copy constructors. The tag parameter referes to the data tag corresponding to the property of interest.

oemolthreadbase

See also

The *Input and Output Threads* section.

Warning

The interface provided here is for advanced users. Most users should be able to accomplish their goals through the high-level *OEReadMolecule* and *OEWriteMolecule* functions.

```
class oemolthreadbase : public oemolstreambase
```

The following methods are publicly inherited from *oemolstreambase*:

ClearCSVHeader	*GetFlavor*	*SetFormat*
GetBinaryIOHandler	*GetFormat*	*close*
GetCSVHeader	*Getgz*	*open*
GetFileName	*SetFlavor*	

The following classes derive from this class:

- *oemolithread*
- *oemolothread*

operator bool

```
operator bool() const
```

GetMol

```
const std::string* GetMol()
```

Retrieve the next molecule in the stream as a `std::string` blocking if there are no molecules currently available. The method will return a zero-pointer (`NULL`) if there are no more molecules left to retrieve.

The user is expected to call `delete` on the `std::string` pointer returned from this method.

> **Note**
>
> This method was specifically designed to be thread safe so multiple threads can call this method on the same *oemolthreadbase* object simultaneously.

PeekMol

```
const std::string* PeekMol()
```

Return the next molecule `std::string` pointer that will be returned by *oemolthreadbase.GetMol* method.

> **Note**
>
> If multiple threads are calling *oemolthreadbase.GetMol* and *oemolthreadbase.PeekMol* then it can not be guaranteed that the thread that called *oemolthreadbase.PeekMol* will get the same pointer when it later calls *oemolthreadbase.GetMol*.

PutMol

```
bool PutMol(const std::string* data)
```

Place a molecule as a `std::string` into the stream blocking if the in-memory buffer is full.

In the case of the *oemolothread* object the output thread will call `delete` on the pointer to free the memory. Passing a zero-pointer (`NULL`) will instruct the *oemolothread* output thread to shut down.

> **Note**
>
> This method was specifically designed to be thread safe so multiple threads can call this method on the same *oemolthreadbase* object simultaneously.

OEMutableCoords

> **Warning**
>
> This class is only usable from the OpenEye C++ toolkits.

```
template<class T>
class OEMutableCoords : public detail::OECoordsBase<T, OEConfBase>
```

This class is used to provide convenient direct access to the memory storing coordinates for a *OEConfBase* through the *OEConfBase.GetCoordsPtr* methods. If the data type of `T` does not match the implementation of *OEConfBase*, this class makes a copy of the coordinates with *OEConfBase.GetCoords* instead. If a copy of the coordinates are made with *OEConfBase.GetCoords*, the destructor of this class will copy the changes made back into the conformer with *OEConfBase.SetCoords*.

> **Note**
>
> This class can incur unnecessary copies if T does not match the implementation of the *OEConfBase* and no modifications were actually made. Use *OEConstCoords* whenever read-only access to coordinates is acceptable.

Constructors

```
OEMutableCoords(OEConfBase* conf)
```

Construct a mutable view of the coordinates in the conformer `conf`.

GetPtr

```
T* GetPtr()
```

Returns a pointer of type `T` to the coordinates of the conformer used in the constructor.

OENbrs

```
struct OENbrs
```

This class represents *OENbrs* that stores an atom pair and the distance between them.

> **See also**
>
> - *OEGetNearestNbrs* function
> - *OENearestNbrs* class

Constructors

```
OENbrs(const OEAtomBase* bgn, const OEAtomBase* end, double dist2)
```

Creates an *OENbrs* object.

GetBgn

```
const OEAtomBase* GetBgn() const
```

Returns the ‘bgn’ atom.

GetDist2

```
double GetDist2() const
```

Returns the square of the distance between the ‘bgn’ and ‘end’ atoms.

GetDist

```
double GetDist() const
```

Returns the distance between the 'bgn' and 'end' atoms.

GetEnd

```
const OEAtomBase* GetEnd() const
```

Returns the 'end' atom.

OENearestNbrs

```
class OENearestNbrs
```

This class represents *OENearestNbrs*.

Constructors

```
OENearestNbrs(const OEMolBase& mol, double maxdist,
              unsigned int method = OENearestNbrsMethod::Default)
```

Creates an *OENearestNbrs* object for nearest neighbor search.

method
: Determines the algorithm that is used to find the nearest neighbors. This value has to be from the *OENearestNbrsMethod* namespace.

GetNbrs

```
OESystem::OEIterBase<OENbrs>* GetNbrs(const float* coords) const
```

Returns an iterator over all atom pairs of the molecule that are closer to each other than the **maxdist** limit. When calculating the distances the coordinates passed as a parameter are utilized rather then the coordinates stored in the molecule of which the *OENearestNbrs* was initialized.

```
OESystem::OEIterBase<OENbrs>* GetNbrs(const OEMolBase& mol) const
```

Returns an iterator over all atoms that are closer to the given atom than the **maxdist** limit.

Returns an iterator over all atom pairs of the two molecules that are closer to each other than the **maxdist** limit. This function calculates **intra-molecular** distances.

```
OESystem::OEIterBase<OENbrs>* GetNbrs(const OEAtomBase* atom) const
```

Returns an iterator over all atoms that are closer to the given atom than the **maxdist** limit.

See also

- *OEGetNearestNbrs* function
- *OENbrs* class

OENthAtom

```
class OENthAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OENthAtom* functor that identifies the Nth atom (*OEAtomBase*) is the molecule.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OENthAtom(unsigned int n, unsigned int start = 0)
```

Constructs the functor with the Nth atom specified by 'n' and the starting atom specified by 'start'.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if iteration over the atoms in the molecule makes the 'atom' the Nth atom in the molecule starting with the first atom being the 'start' atom.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OENthAtom* object is dynamically allocated and owned by the caller.

OEOmegaConfTest

```
class OEOmegaConfTest : public OEConfTestBase
```

This implementation of *OEConfTestBase* combines subsequent connection tables into a multi-conformer molecule if they:

- have the same number of atoms and bonds in the same order
- have atoms and bonds with identical properties in the same order
- have the same atom and bond stereochemistry, except for invertible nitrogen stereochemistry
- have the same title (optional)

OEOmegaConfTest shouldn't be used for reading multi-conformer molecules in PDB or XYZ file format. These file formats typically require the use of *OEDetermineConnectivity* (and *OEPerceiveBondOrders*) which may potentially perceive and create bonds in different orders (or arbitrarily assign bond orders and Kekulé forms differently) for different conformers of the same molecule. This disrupts the ordering (and property) constraints described above.

Generally, file formats other than PDB and XYZ should be used for storing multi-conformer molecules (as consecutive connection tables).

The following methods are publicly inherited from *OEConfTestBase*:

CombineMols	*CompareMols*	*CreateCopy*

Constructors

```
OEOmegaConfTest(bool compTitles = true)
```

Constructs an *OEOmegaConfTest* object. The argument 'compTitle' specifies whether to compare molecule titles (accessed by *OEMolBase.GetTitle*). If the constructor is called with no arguments or with the argument `true`, molecule titles will be required to be the same. Otherwise, the titles will not be compared. In the latter instance, each conformer will have the individual title of its original connection table and the multi-conformer molecule will reflect the title of the active conformer.

CombineMols

```
bool CombineMols(OEMCMolBase& m1, OEMolBase& m2)
```

CompareMols

```
bool CompareMols(OEMolBase& m1, OEMolBase& m2) const
```

Returns whether the two *OEMolBase* objects are the same by checking the four conditions listed above.

CreateCopy

```
OEConfTestBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEOmegaConfTest* object is dynamically allocated and owned by the caller.

OEOutFileStringParameter

```
class OEOutFileStringParameter : public OEFileStringParameter
```

The *OEOutFileStringParameter* represents parameter that has `string` value representing a writable file format for modeling objects, such as a molecule or a design unit.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*

- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEOutFileStringParameter()
OEOutFileStringParameter(const std::string& name,
                         const unsigned fileType,
                         const unsigned fileTypeFlavor = 0u)
OEOutFileStringParameter(const std::string& name,
                         const std::string& fileExt)
OEOutFileStringParameter(const OEOutFileStringParameter&)
```

Default and copy constructors.

Constructs an *OEOutFileStringParameter* instance using the specified set of parameters. The `fileType` argument refers to constants defined in the *OEFileStringType* namespace. The `fileTypeFlavor` argument refers to constants defined in the *OEFileStringTypeFlavor* namespace.

OEParseSmilesOptions

```
class OEParseSmilesOptions
```

An class for handling options for the *OEParseSmiles* function.

Constructors

```
OEParseSmilesOptions(bool canon = false, bool strict = false,
                     bool quiet = false, bool cxsmiles = false)
```

Initializes the object with the specified options.

GetCanon

```
bool GetCanon() const
```

A value of `true` specifies that the *OEKekulize* step can be skipped. Defaults to `false`.

GetCXSMILES

```
bool GetCXSMILES() const
```

A value of `true` specifies that a CXSMILES formatted appendix is allowed to be present in the parse string, and extra checking will be employed to capture any enhanced stereogroup information therein. A value of `false` results in a CXSMILES appendix being ignored and retained as part of the molecule title. Defaults to `false`.

> **Warning**
>
> Unparsed CXSMILES appendix information captured in the molecule title will generally be corrupt after regeneration of SMILES information.

GetStrict

```
bool GetStrict() const
```

A value of `true` specifies that the *OEParseSmiles* will apply more rigorous sanity checks on the parsed SMILES string. Defaults to `false`.

GetQuiet

```
bool GetQuiet() const
```

A value of `true` specifies that there will be no output written to *OEThrow* if *OEParseSmiles* fails to parse the string. Defaults to `false`.

SetCanon

```
void SetCanon(const bool)
```

This setting can be used to circumvent the post-processing kekulization test of *OEParseSmiles*. Passing a boolean `true` value to this argument indicates to the parser that the SMILES string should be assumed to be well-formed and the usual kekulization (by calling *OEKekulize*) step may be omitted. This can be used to speed-up parsing of a large database, but has the side-effect that bond orders are not correctly assigned for aromatic molecules.

SetCXSMILES

```
void SetCXSMILES(const bool)
```

Setting a value of `true` specifies that a CXSMILES formatted appendix is allowed to be present in the parse string, and extra checking should be employed to capture any enhanced stereogroup information therein. Setting a value of `false` results in a CXSMILES appendix being ignored and captured as part of the molecule title.

SetStrict

```
void SetStrict(const bool)
```

This setting controls whether *OEParseSmiles* should operate in strict mode. By default, the SMILES parser attempts to process any reasonably formed SMILES string. If this setting is `true`, the parser applies more rigorous sanity checking. For example, the SMILES `C==C` is accepted by the default non-strict parser, but rejected by the strict parser.

SetQuiet

```
void SetQuiet(const bool)
```

This setting can be used to force *OEParseSmiles* to not write anything to *OEThrow* in the case of a failure. This can be useful for programs that want to determine if a string is a valid SMILES string without spewing unnecessary logging information to the terminal.

OEPartPredT

```
template<class T>
class OEPartPredT : public OESystem::OEUnaryPredicate<T>
```

This class represents *OEPartPredT* template functor that holds multiple parts of a molecule and identifies objects (that provide `GetIdx` method such as *OEAtomBase*, *OEBondBase*) that belong to different part of a molecule based on the state of the functor.

See also

- *Molecule Partitioning* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following specialization exists for this template:

Specialized Functor	Definition
OEPartPredAtom	*OEPartPredT* <*OEAtomBase*>

Constructors

```
OEPartPredT(unsigned int* parts, unsigned int size)
```

Constructs the functor where the parts are specified in the passed integer array ('parts'), the size of the 'parts' array is is specified by the 'size' argument.

If the *OEPartPredT* is used to identify atoms (*OEAtomBase*), then 'parts' array is assumed to be of length equal to the number return by the *OEMolBase.GetMaxAtomIdx* method, where parts[atom. *GetIdx()*] is the integer associated with the part to which atom belongs.

```
OEPartPredT(const OEPartPredT& rhs)
```

Copy constructor.

operator()

```
bool operator()(const T& t) const
```

If the *OEPartPredT* is used to identify atoms, then this method returns `true`, if parts[atom. *GetIdx()*] == part, where 'parts' is the array passed on construction, and 'part' is the argument set by the *OEPartPredT.SelectPart* method.

CreateCopy

```
OESystem::OEUnaryFunction<T, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEPartPredT* object is dynamically allocated and owned by the caller.

SelectPart

```
void SelectPart(unsigned int part)
```

Sets the 'part' argument that is used to select a specific part of a molecule.

OEPredicateParameter

```
template<class TT>
template <class TT>class OEPredicateParameter : public OESystem::OEPointerParameter<TT>
```

The *OEPredicateParameter* represents parameter that has value of predicate type.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

The *OEPredicateParameter* class defines the following public methods:

- *GetDefault* and *SetDefault*
- *GetSetting*
- *GetValue* and *SetValue*

The following classes derive from this class:

- *OEUnAtomPredParameter*
- *OEUnBondPredParameter*

Constructors

```
OEPredicateParameter()
OEPredicateParameter(const std::string& name)
OEPredicateParameter(const OEPredicateParameter&)
```

Default and copy constructors.

Constructs an *OEPredicateParameter* instance using the specified set of parameters.

GetDefault

```
TT* GetDefault() const
```

Returns the current default value of the parameter.

GetSetting

```
TT* GetSetting() const
```

Returns the current setting of the parameter.

GetValue

```
TT* GetValue() const
```

Returns the current user-defined value of the parameter.

SetDefault

```
bool SetDefault(const TT& value)
```

Sets the default value of the parameter.

SetValue

```
bool SetValue(const TT& value)
```

Sets the user-defined value of the parameter.

OEQAtomBase

```
class OEQAtomBase : public OEAtomBase , public OEQBase
```

The *OEQAtomBase* class is the abstract interface for representing atoms and atom expressions within **OpenEye's Toolkits**. For a description of query expressions and their utility in graph matching refer to the *OEExprOpts Namespace* section. An *OEQAtomBase* object is created by calling the *OEMolBase.NewAtom* method on the parent molecule.

The following methods are publicly inherited from *OEAtomBase*:

operator=	*GetRadius*	*IsSulfur*
AddBond	*GetRxnRole*	*OrderBonds*
Clear	*GetStereo*	*RemoveBond*
GetAtomicNum	*GetSymmetryClass*	*SetAromatic*
GetAtoms	*GetTotalHCount*	*SetAtomicNum*
GetType		
GetBond	*GetValence*	*SetChiral*
GetBonds	*SetFormalCharge*	
GetDegree	*HasStereoSpecified*	*SetHyb*
GetExplicitDegree	*IsAromatic*	
GetExplicitHCount	*IsCarbon*	*SetImplicitHCount*
GetExplicitValence	*IsChiral*	*SetInRing*
GetFormalCharge	*IsConnected*	*SetIntType*
GetHvyDegree	*IsDataType*	*SetIsotope*
GetHvyValence	*IsDeleted*	*SetMapIdx*
GetHyb	*IsHalogen*	*SetName*
GetIdx	*IsHydrogen*	*SetPartialCharge*
GetImplicitHCount	*IsInRing*	*SetRadius*
GetIntType	*IsMetal*	*SetRxnRole*
GetIsotope	*IsNitrogen*	*SetStereo*
GetMapIdx	*IsOxygen*	*SetSymmetryClass*
GetName	*IsPhosphorus*	*SetType*
GetParent	*IsPolar*	*Sweep*
GetPartialCharge	*IsPolarHydrogen*	

The following methods are publicly inherited from *OEQBase*:

GetExpr	*SetExpr*

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

GetParentAtom

```
OEAtomBase* GetParentAtom() const = 0
```

GetQAtoms

```
OESystem::OEIterBase<OEQAtomBase>* GetQAtoms() const = 0
```

Returns an iterator over the neighboring query atoms of this atom. The returned iterator traverses only the explicit atoms that are bonded to the *OEQAtomBase*. The number of neighbors in the iterator is identical to *OEAtomBase.GetExplicitDegree*.

```
OESystem::OEIterBase<OEQAtomBase>* GetQAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase> &) const = 0
```

Returns an iterator over the neighboring query atoms of the *OEQAtomBase* that match the predicate (*OEUnaryPredicate*)

GetQBonds

```
OESystem::OEIterBase<OEQBondBase>* GetQBonds() const = 0
```

Returns an iterator over the query bonds connected to the atom. The returned iterator traverses only the explicit bonds that are attached to the *OEQAtomBase*. The number of bonds in the iterator is identical to *OEAtomBase.GetExplicitDegree*.

```
OESystem::OEIterBase<OEQBondBase>* GetQBonds(const OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase> &) const = 0
```

Returns an iterator over the query bonds of the *OEQAtomBase* that match the predicate (*OEUnaryPredicate*)

IsDataType

```
bool IsDataType(const void *) const
```

OEQBase

```
class OEQBase
```

This class represents *OEQBase*.

The *OEQBase* is the abstract class which defines the API for storage and retrieval of expressions used in graph matching operations.

The following classes derive from this class:

- *OEQAtomBase*
- *OEQBondBase*

GetExpr

```
const OEExprBase* GetExpr() const = 0
OEExprBase* GetExpr() =0
```

This pure virtual method defines the interface for retrieving a stored expressions from a *OEQBase* parent class instance. If no expression was stored previously a NULL pointer may be returned. The pointer returned from this method should therefore be checked before use.

SetExpr

```
bool SetExpr(const OEExprBase *)=0
bool SetExpr(const OEExprBase &)=0
```

This pure virtual method defines the interface for storage of an expression. The parent class instance takes ownership of the memory associated with the expression pointer. Any existing expressions stored in the parent instance will be destroyed. Stored expressions are destroyed along with their parent *OEQBase* object.

OEQBondBase

```
class OEQBondBase : public OEBondBase , public OEQBase
```

The *OEQBondBase* class is the abstract interface for representing bonds and bond matching expressions within **OpenEye's Toolkits**. For a description of query expressions and their utility in graph matching refer to the *OEExprOpts Namespace* section. An *OEQBondBase* object is created by calling the *OEMolBase.NewBond* method on the parent molecule.

The following methods are publicly inherited from *OEBondBase*:

operator=	*GetStereo*	*SetChiral*
Clear	*GetType*	*SetEnd*
GetBgn	*HasStereoSpecified*	
GetBgnIdx	*IsAromatic*	*SetInRing*
GetEnd	*IsChiral*	*SetIntType*
GetEndIdx	*IsDataType*	*SetOrder*
GetIdx	*IsDeleted*	*SetStereo*
GetIntType	*IsInRing*	*SetType*
GetNbr	*IsRotor*	*SwapEnds*
GetOrder	*SetAromatic*	*Sweep*
GetParent	*SetBgn*	

The following methods are publicly inherited from *OEQBase*:

GetExpr	*SetExpr*

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

GetParentBond

```
OEBondBase* GetParentBond() const = 0
```

IsDataType

```
bool IsDataType(const void *) const
```

OEQMol

```
class OEQMol
```

The *OEQMol* is a specific implementation of query molecules.

The following methods are publicly inherited from *OEMolBase*:

operator=	*GetDimension*	*NumBonds*
operator bool	*GetEnergy*	*NumGroups*
Clear	*GetGroup*	*OrderAtoms*
ClearCoords	*GetGroups*	*OrderBonds*
Compress	*GetMaxAtomIdx*	*ResetPerceived*
Count	*GetMaxBondIdx*	*SetCoords*
CreateCopy	*GetTitle*	*SetDimension*
DeleteAtom	*HasPerceived*	*SetEnergy*
DeleteBond	*IsDataType*	*SetPerceived*
DeleteGroup	*IsDeleted*	*SetRxn*
GetAtom	*IsRxn*	*SetTitle*
GetAtoms	*NewAtom*	*Sweep*
GetBond	*NewBond*	*UnCompress*
GetBonds	*NewGroup*	
GetCoords	*NumAtoms*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

All query molecule constructors provide the ability to specify the implementation type through the 'type' argument. The default implementation type (*OEQMolType.OEDefault*) is provided as a default argument when not explicitly specified. If additional query molecule implementations exist they will be listed in the *OEQMolType* namespace.

> **Note**
>
> Copy constructors which take a non-query molecule type as an argument do not, by default, build atom and bond expressions used in graph matching. The *OEQMolBase.BuildExpressions* method should be called to build query expressions after copy constructing a molecule using something other than a query molecule.

```
explicit OEQMol(unsigned int type = OEQMolType::OEDefault)
```

Default constructor.

```
OEQMol(const OEQMol &)
```

Copy constructor.

```
OEQMol(const OEQMolBase &)
explicit OEQMol(const OEQMolBase *)
OEQMol(const OEQMol &, unsigned int type)
```

Construct a query molecule from a query molecule (*OEQMol*). Atom and bond expressions will be copied from the source molecule.

```
OEQMol(const OEQMolBase &, unsigned int type)
```

Construct a query molecule from a query base molecule (*OEQMolBase*). Atom and bond expressions will be copied from the source molecule.

```
OEQMol(const OEMol &, unsigned int type = OEQMolType::OEDefault)
```

Construct a query molecule from a multi-conformer molecule (*OEMol*).

```
OEQMol(const OEMolBase &, unsigned int type = OEQMolType::OEDefault)
```

Construct a query molecule from a base molecule (*OEMolBase*).

```
OEQMol(const OEGraphMol &,
       unsigned int type = OEQMolType::OEDefault)
```

Construct a query molecule from a graph molecule (*OEGraphMol*).

```
OEQMol(const OEMCMolBase &,
       unsigned int type = OEQMolType::OEDefault)
```

Construct a query molecule from a multi-conformer base molecule (*OEMCMolBase*).

BuildExpressions

```
bool BuildExpressions(const OEMatchPair<OEExprBase>& mp)
bool BuildExpressions(unsigned int atomopts, unsigned int bondopts)
```

GetQAtoms

```
OESystem::OEIterBase<OEQAtomBase>* GetQAtoms() const
OESystem::OEIterBase<OEQAtomBase>* GetQAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase>& pred) const
```

GetQBonds

```
OESystem::OEIterBase<OEQBondBase>* GetQBonds() const
OESystem::OEIterBase<OEQBondBase>* GetQBonds(const OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase>& pred) const
```

QMol

```
const OEQMolBase& QMol() const
OEQMolBase& QMol()
```

Returns a query molecule (*OEQMol*) as an *OEQMolBase* reference. When passing an *OEQMol* object to a function which is overloaded to take either an *OEMolBase* or *OEQMolBase* object it is necessary to explicitly specify which overloaded method is desired. This method is equivalent to the cast operator *OEQMol.operator OEQMolBase &*.

SCMol

```
const OEMolBase& SCMol() const
OEMolBase& SCMol()
```

Returns a query molecule (*OEQMol*) as an *OEMolBase* reference. When passing an *OEQMol* to a function which is overloaded to take either an *OEMolBase* or *OEQMolBase* object it is necessary to explicitly specify which overloaded method is desired. This method is equivalent to the cast operator *OEQMol.operator OEMolBase &*.

OEQMolBase

```
class OEQMolBase : public OEMolBase
```

This class represents *OEQMolBase*.

The *OEQMolBase* is an abstract class that extends *OEMolBase* and provides facilities useful in graph matching. Query molecules are used in matching operations such as substructure search (*OESubSearch*), clique detection (*OECliqueSearch*), and maximum common substructure search (*OEMCSSearch*). Query expressions (*OEExprBase*) stored in query atoms (*OEQAtomBase*) and query bonds (*OEQBondBase*) are used to represent the boolean logic used in graph matching operations. Query molecules (*OEQMolBase*) conveniently collect query atoms and query bonds into a molecule suitable for use as a query graph in matching operations. As a derived type of *OEMolBase*, query molecules are valid arguments for functions which take a base molecule class as an argument.

The following methods are publicly inherited from *OEMolBase*:

operator=	*GetDimension*	*NumBonds*
operator bool	*GetEnergy*	*NumGroups*
Clear	*GetGroup*	*OrderAtoms*
ClearCoords	*GetGroups*	*OrderBonds*
Compress	*GetMaxAtomIdx*	*ResetPerceived*
Count	*GetMaxBondIdx*	*SetCoords*
CreateCopy	*GetTitle*	*SetDimension*
DeleteAtom	*HasPerceived*	*SetEnergy*
DeleteBond	*IsDataType*	*SetPerceived*
DeleteGroup	*IsDeleted*	*SetRxn*
GetAtom	*IsRxn*	*SetTitle*
GetAtoms	*NewAtom*	*Sweep*
GetBond	*NewBond*	*UnCompress*
GetBonds	*NewGroup*	
GetCoords	*NumAtoms*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEQMolBase()
OEQMolBase(const OEMolBase& m)
OEQMolBase(const OEQMolBase& m)
```

Default and copy constructors.

BuildExpressions

```
bool BuildExpressions(const OEMatchPair<OEExprBase> &)
```

```
bool BuildExpressions(unsigned int atomopts, unsigned int bondopts)
```

Builds query expressions (*OEExprBase*) for the atoms (*OEQAtomBase*) and bonds (*OEQBondBase*) contained in a query molecule object. Any pre-existing query expressions are destroyed by this method. The bits in the unsigned integers passed to the method define how query expressions will be built based on the attributes of each atom and bond instance. The bitwise options are stored in the *OEExprOpts* namespace. This method provides the ability to convert attributes of base atoms and bonds to query expressions that are consistent with the semantics desired for a graph match procedure. Each requested expression is constructed, using data from the parent atom or bond if necessary. During a graph match procedure, the resulting expression composition will be used to evaluate atom and bond graph component equivalences.

> **Note**
>
> *OEQMolBase.BuildExpressions* uses the data present in a query molecule only when it is called. Subsequent modifications to atom and bond data to not cause the expressions to be updated to reflect changes. An additional call to *OEQMolBase.BuildExpressions* must be made to synchronize expressions with atom and bond data.

CreateCopy

```
OESystem::OEBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEQMolBase* object is dynamically allocated and owned by the caller.

GetQAtoms

```
OESystem::OEIterBase<OEQAtomBase>* GetQAtoms() const = 0
```

Returns an iterator over all the query atoms (*OEQAtomBase*) of a query molecule. By default, this returns all the query atoms of the molecule in the order they were created. The ordering of the atoms returned by *OEQMolBase.GetQAtoms* may be modified by *OEMolBase.OrderAtoms*.

```
OESystem::OEIterBase<OEQAtomBase>* GetQAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase> &) const = 0
```

Returns an iterator over all of the query atoms of a query molecule that match the specified atom predicate (*OEUnaryPredicate*).

GetQBonds

```
OESystem::OEIterBase<OEQBondBase>* GetQBonds() const = 0
```

Returns an iterator over all the query bonds (*OEQBondBase*) of a query molecule. By default, this returns all the query bonds of the molecule in the order they were created. The ordering of the bonds returned by *OEQMolBase.GetQBonds* may be modified by *OEMolBase.OrderBonds*.

```
OESystem::OEIterBase<OEQBondBase>* GetQBonds(const OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase> &) const = 0
```

Returns an iterator over all of the query bonds of a query molecule that match the specified bond predicate (*OEUnaryPredicate*).

IsDataType

```
bool IsDataType(const void *) const
```

OEQuaternion

```
class OEQuaternion : public OETransBase
```

This class represents *OEQuaternion*.

The *OEQuaternion* class performs quaternion rotation transformation.

The following methods are publicly inherited from *OETransBase*:

CreateCopy	*GetInverse*
Extract	*Transform*

Constructors

```
OEQuaternion()
OEQuaternion(const float* quat)
OEQuaternion(const double* quat)
OEQuaternion(const float* vec, float angle)
OEQuaternion(const double* vec, double angle)
OEQuaternion(const double w, const double x, const double y,
             const double z)
OEQuaternion(const float w, const float x, const float y,
             const float z)
```

Default and copy constructors.

CreateCopy

```
OETransBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEQuaternion* object is dynamically allocated and owned by the caller.

Extract

```
void Extract(float* r, float* t) const
void Extract(float* r, double* t) const
void Extract(double* r, float* t) const
void Extract(double* r, double* t) const
```

GetInverse

```
OETransBase* GetInverse() const
```

GetQuaternion

```
void GetQuaternion(float* quat) const
void GetQuaternion(double* quat) const
```

OERefInputAppOptions

```
class OERefInputAppOptions : public OESystem::OEOptions
```

The *OERefInputAppOptions* represents an options class, representing an applications interface that requires an additional reference or target input, along with an input reader and an output writer file.

See also

- *OEOptions* class

Code Examples

- Docking Molecules example
- Generating Poses with POSIT example

The *OERefInputAppOptions* class defines the following public methods:

- *GetInFile* and *SetInFile*
- *GetOutFile* and *SetOutFile*
- *GetRefFile* and *SetRefFile*

Constructors

```
OERefInputAppOptions(const OESystem::OEOptions& opts,
                     const std::string& name,
                     const unsigned inFileType,
                     const unsigned outFileType,
                     const unsigned refFileType,
                     const std::string& refCmdName)
OERefInputAppOptions(const OESystem::OEOptions& opts,
                     const std::string& name,
                     const unsigned inFileType,
                     const unsigned outFileType,
                     const std::string& refFileExt,
                     const std::string& refCmdName)
OERefInputAppOptions(const OESystem::OEOptions& opts,
                     const std::string& name,
                     const unsigned inFileType,
                     const unsigned refFileType,
                     const std::string& refCmdName)
OERefInputAppOptions(const OESystem::OEOptions& opts,
                     const std::string& name,
                     const unsigned inFileType,
                     const std::string& refFileExt,
                     const std::string& refCmdName)
OERefInputAppOptions(const OERefInputAppOptions &)
```

Default and copy constructors.

The `opts` represents the sub-options that should be wrapped to extend this options to generate the API for the intended application, the `name` represents the name of the application, and the `refCmdName` represents the command line flag for reference input. The `inFileType`, `outFileType`, and `refFileType` refers to if the input, output and the reference input file types as defined in the *OEFileStringType* namespace. The second overload accepts the reference input file type as a string. Variation of constructors that does not require an output file can be used when an output is not desired in an application.

GetInFile

```
std::string GetInFile() const
```

See *SetInFile* method.

GetOutFile

```
std::string GetOutFile() const
```

See *SetOutFile* method.

GetRefFile

```
std::string GetRefFile() const
```

See *SetRefFile* method.

SetInFile

```
bool SetInFile(string)
```

Sets name of the input file for the application. **Default: None**

SetOutFile

```
bool SetOutFile(string)
```

Sets name of the output file for the application. **Default: None**

SetRefFile

```
bool SetRefFile(string)
```

Sets name of the reference/target input file for the application. **Default: None**

OEResidue

```
class OEResidue
```

The *OEResidue* class is used to attach biopolymer information to each *OEAtomBase* object. This class contains a number of fields specific to processing macromolecules, such as proteins and nucleic acids. Each atom may be annotated with a unique *OEResidue*, hence *OEResidue* objects are not shared between different atoms. If the fields of an *OEResidue* associated with a single atom are updated, only that atom is affected. To adapt to the change for mmCIF where a chainID can now contain multiple characters, the chainID is now stored as an string. The existing GetChainID() function will return just the first character of the chain ID. We encourage users to switch to use GetExtChainID() which will return the string.

See also

- *OEHasResidue* function

- *OEAtomGetResidue* function
- *OEAtomSetResidue* function

Note

See section *Biopolymer Residues* for a detailed description of the data stored in an *OEResidue*.

Constructors

```
OEResidue()
```

Default constructor.

```
OEResidue(const OEResidue& rhs)
```

Copy constructor.

operator bool

```
operator bool() const
```

Determines whether any field of an *OEResidue* has a non-default value.

GetAlternateLocation

```
char GetAlternateLocation() const
```

Returns the 'alternate location' property of an atom in a residue. The 'alternate location' property may be set using the *OEResidue.SetAlternateLocation* method.

GetBFactor

```
float GetBFactor() const
```

Returns the crystallographic 'b-factor' property of an atom in a residue. The 'b-factor' property may be set using the *OEResidue.SetBFactor* method.

See also

- *OEHasAnisou*
- *OEGetAnisou*
- *OESetAnisou*

GetChainID

```
char GetChainID() const
```

This method is deprecated and should no longer be used. Instead, use *OEResidue.GetExtChainID*. Returns the first character of the 'chain identifier' property of the residue. The 'chain identifier' property may be set using the *OEResidue.SetExtChainID* method.

GetExtChainID

```
string GetExtChainID() const
```

Returns the 'chain identifier' property of the residue. The 'chain identifier' property may be set using the *OEResidue.SetExtChainID* method. If reading from mmCIF, this property is `auth_asym_id` field from the mmCIF spec.

GetSubChainID

```
string GetSubChainID() const
```

Returns the 'subchain identifier' property of the residue. The 'subchain identifier' property may be set using the *OEResidue.SetSubChainID* method. When reading from PDB, this is default to a space character. When reading from mmCIF, these are now a further subdivision of the chain identifier based on connected components. If reading from mmCIF, this property is `label_asym_id` field from the mmCIF spec.

GetFragmentNumber

```
int GetFragmentNumber() const
```

Returns the 'fragment number' property of a residue. Fragment numbers are integer indices that define connected components of a macromolecule. When reading from a PDB file, the first atom is placed in fragment number 1. Each time a TER, END or ENDM record is encountered within a single connection table, the fragment number is incremented. Similarly, when writing a PDB file, a TER record is written between any consecutive pair of atoms with different fragment numbers. The fragment number property may be set using the *OEResidue.SetFragmentNumber* number property.

GetInsertCode

```
char GetInsertCode() const
```

Returns the 'insertion code' of a residue. Insertion codes are single character suffixes used to distinguish insertions when using a standardized residue numbering system. By default, this property contains a single space character, `' '`. The insertion code property of a residue may be set using the *OEResidue.SetInsertCode* method.

GetModelNumber

```
int GetModelNumber() const
```

Returns the 'model number' property of a residue. For NMR or molecular dynamics files, the model number contains the index of the NMR model or dynamics time-step within a file. When reading from mmCIF, the model numbers start from 1. The 'model number' property may be set using the *OEResidue.SetModelNumber* method.

GetName

```
const char* GetName() const
```

Returns the 'residue name' property of a residue. The 'residue name' property may be set using the *OEResidue.SetName* method.

GetOccupancy

```
float GetOccupancy() const
```

Returns the crystallographic 'occupancy' property of an atom in a residue. The default value is `1.0`. The 'occupancy' property may be set using the *OEResidue.SetOccupancy* method.

GetResidueNumber

```
int GetResidueNumber() const
```

Returns the 'residue number' property of a residue. The 'residue number' property may be set using the *OEResidue.SetResidueNumber* method.

GetEntityID

```
string GetEntityID() const
```

Returns the 'entity ID' property of a residue. The 'entity ID' property may be set using the *OEResidue.SetEntityID* method.

GetSequenceID

```
int GetSequenceID() const
```

Returns the 'sequence id' property of a residue, if set by reading an mmCIF file. The 'sequence id' property may be set using the *OEResidue.SetSequenceID* method. The number of typically sequential starting from 1, contrary to the 'residue number' which typically follows the biological sequence numbering. If reading from mmCIF, this property is `label_seq_id` field from the mmCIF spec.

GetSecondaryStructure

```
int GetSecondaryStructure() const
```

Returns the protein 'secondary structure' property of a residue. The default value is zero. The 'secondary structure' property of a residue may be set using the *OEResidue.SetSecondaryStructure* method.

GetSerialNumber

```
int GetSerialNumber() const
```

Returns the 'atom serial number' property of an atom in a residue. The 'atom serial number' property may be set using the *OEResidue.SetSerialNumber* method.

IsHetAtom

```
bool IsHetAtom() const
```

Returns the 'hetero atom' property of an atom in a residue. The 'hetero atom' property may be set using the *OEResidue.SetHetAtom* method.

Note

A 'hetero atom' is distinguished from a 'standard atom' in a PDB file by using the record type `HETATM` rather than `ATOM`.

SetAlternateLocation

```
bool SetAlternateLocation(char v)
```

Sets the 'alternate location' property of an atom in a residue. The default value is a space character. The 'alternate location' property of an atom in a residue may be retrieved using the *OEResidue.GetAlternateLocation* method.

SetBFactor

```
bool SetBFactor(float v)
```

Sets the crystallographic 'b-factor' property of an atom in a residue. The default value is `0.0`. The 'b-factor' property of an atom in a residue may be retrieved using the *OEResidue.GetBFactor* method.

See also

- *OEHasAnisou*
- *OESetAnisou*
- *OEGetAnisou*

SetChainID

```
bool SetChainID(char v)
```

This method is deprecated and should no longer be used. Instead, use *OEResidue.SetExtChainID*. Sets the 'chain identifier' property of a residue. This sets the same property as *OEResidue.SetExtChainID* The default value is a space character. The 'chain identifier' property of a residue may be retrieved using the *OEResidue.GetExtChainID* method.

SetExtChainID

```
bool SetExtChainID(const string &)
```

Sets the 'chain identifier' property of a residue. The default value is a space character. The 'chain identifier' property of a residue may be retrieved using the *OEResidue.GetExtChainID* method.

SetSubChainID

```
bool SetSubChainID(const string &)
```

Sets the 'subchain identifier' property of a residue. The default value is a space character. The 'subchain identifier' property of a residue may be retrieved using the *OEResidue.GetSubChainID* method.

SetFragmentNumber

```
bool SetFragmentNumber(int v)
```

Sets the fragment number property of a residue. Fragment numbers are integer indices that define the connected components of a macromolecule. When reading from a PDB file, the first atom is placed in fragment number 1. Each time a TER, END or ENDM record is encountered within a single connection table, the fragment number is incremented. Similarly, when writing a PDB file, a TER record is written between any consecutive pair of atoms with different fragment numbers. When reading from an mmCIF file, this number is read directly from the input as it is defined in that file spec. The default value is zero. The 'fragment number' property of a residue may be retrieved using the *OEResidue.GetFragmentNumber* method.

SetHetAtom

```
bool SetHetAtom(bool v)
```

Set the 'hetero atom' property of an atom in a residue. The default value is `false`. The 'hetero atom' property of an atom in a residue may be retrieved using the *OEResidue.IsHetAtom* method.

SetInsertCode

```
bool SetInsertCode(char v)
```

Sets the 'insertion code' of a residue. Insertion codes are single character suffices used to distinguish insertions when using a standardized residue numbering system. By default, this property contains a single space character `' '`. The 'insertion code' property of a residue may be retrieved using the *OEResidue.GetInsertCode* method.

SetModelNumber

```
bool SetModelNumber(int v)
```

Sets the *model number* property of a residue. For NMR or molecular dynamics files, the model number contains the index of the NMR model or dynamics time-step within a file. The default value is 0. The 'model number' property of a residue may be retrieved using the *OEResidue.GetModelNumber* method.

SetName

```
bool SetName(const char* v)
bool SetName(const std::string& v)
```

Sets the 'residue name' property of a residue. The default value is `MOL`. The 'residue name' property of a residue may be retrieved using the *OEResidue.GetName* method.

SetOccupancy

```
bool SetOccupancy(float v)
```

Sets the crystallographic 'occupancy' property of an atom in a residue. The default value is `1.0`. The 'occupancy' property of an atom in a molecule may be retrieved using the *OEResidue.GetOccupancy* method.

SetResidueNumber

```
bool SetResidueNumber(int v)
```

Sets the 'residue number' property of a residue. The default value is 1. The 'residue number' property of a residue may be retrieved using the *OEResidue.GetResidueNumber* method.

SetEntityID

```
bool SetEntityID(const string &)
```

Sets the 'Entity ID' property of a residue. The default value is a single space character, `' '`. The 'Entity ID' property of a residue may be retrieved using the *OEResidue.GetEntityID* method.

SetSequenceID

```
bool SetSequenceID(int v)
```

Sets the 'sequence id' property of a residue. The default value is `-1`. The 'sequence id' property of a residue may be retrieved using the *OEResidue.GetSequenceID* method.

SetSecondaryStructure

```
bool SetSecondaryStructure(int v)
```

Sets the protein 'secondary structure' property of a residue. The default value is 0. The 'secondary structure' property of a residue may be retrieved using the *OEResidue.GetSecondaryStructure* method.

SetSerialNumber

```
bool SetSerialNumber(int v)
```

Sets the 'atom serial number' property of an atom in a residue. The default value is 0. The 'atom serial number' property may be retrieved using the *OEResidue.GetSerialNumber* method.

OERotMatrix

```
class OERotMatrix : public OETransBase
```

This class represents *OERotMatrix*.

The *OERotMatrix* class performs rotation specified by a rotation matrix.

The following methods are publicly inherited from *OETransBase*:

CreateCopy	*GetInverse*
Extract	*Transform*

Constructors

```
OERotMatrix()
OERotMatrix(const float* mat)
OERotMatrix(const double* mat)
OERotMatrix(const OERotMatrix& rhs)
```

Default and copy constructors.

CreateCopy

```
OETransBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OERotMatrix* object is dynamically allocated and owned by the caller.

Extract

```
void Extract(float* r, float* t) const
void Extract(double* r, float* t) const
void Extract(float* r, double* t) const
void Extract(double* r, double* t) const
```

GetInverse

```
OETransBase* GetInverse() const
```

GetRotMatrix

```
void GetRotMatrix(float* mat) const
void GetRotMatrix(double* mat) const
```

OESDDataPair

```
class OESDDataPair
```

This class represents *OESDDataPair*.

Constructors

```
OESDDataPair()
OESDDataPair(std::string t, std::string v)
```

Default and copy constructors.

GetTag

```
const char* GetTag() const
```

GetValue

```
const char* GetValue() const
```

SetTag

```
void SetTag(const char* t)
```

SetValue

```
void SetValue(const char* v)
```

OESimilarByRMSD

```
class OESimilarByRMSD : public OESystem::OEBinaryPredicate<OEChem::OEConfBase,␣
↪OEChem::OEConfBase>
```

This class represents *SimilarByRMSD*, a functor that identifies if two conformers (*OEConfBase*) are similar.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

Constructors

```
OESimilarByRMSD(const OESimilarByRMSD&)
OESimilarByRMSD(const double rmsd)
```

Default and copy constructors.

operator()

```
bool operator()(const OEChem::OEConfBase& conf1, const OEChem::OEConfBase& conf2) const
```

Returns `true`, if the two specified conformers are similar within the specified RMSD, i.e., if the RMSD between the two conformers is smaller or equal to the specified value.

CreateCopy

```
OESystem::OEBinaryFunction<OEChem::OEConfBase, OEChem::OEConfBase, bool> * CreateCopy()␣
↪const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OESimilarByRMSD* object is dynamically allocated and owned by the caller.

OESimpleAppOptions

```
class OESimpleAppOptions : public OESystem::OEOptions
```

The *OESimpleAppOptions* represents a simple options class, representing a simple applications interface with an input reader file and an output writer file.

See also

- *OEOptions* class

Code Examples

- Generating Conformers example
- Generating Densely Sampled Conformers example
- Generating Stereoisomers example
- Generating Torsion Driven Conformation Examples example
- Fragment Library Generation Examples example
- Macrocycle Examples example

The *OESimpleAppOptions* class defines the following public methods:

- *GetInFile* and *SetInFile*
- *GetOutFile* and *SetOutFile*

Constructors

```
OESimpleAppOptions(const std::string& name,
                   const unsigned inFileType,
                   const unsigned outFileType)
OESimpleAppOptions(const std::string& name,
                   const std::string& inFileExt,
                   const unsigned outFileType)
```

(continues on next page)

(continued from previous page)

```
OESimpleAppOptions(const std::string& name,
                   const unsigned inFileType,
                   const std::string& outFileExt)
OESimpleAppOptions(const std::string& name,
                   const std::string& inFileExt,
                   const std::string& outFileExt)
OESimpleAppOptions(const OESystem::OEOptions& opts,
                   const std::string& name,
                   const unsigned inFileType,
                   const unsigned outFileType)
OESimpleAppOptions(const OESystem::OEOptions& opts,
                   const std::string& name,
                   const std::string& inFileExt,
                   const unsigned outFileType)
OESimpleAppOptions(const OESystem::OEOptions& opts,
                   const std::string& name,
                   const unsigned inFileType,
                   const std::string& outFileExt)
OESimpleAppOptions(const OESystem::OEOptions& opts,
                   const std::string& name,
                   const std::string& inFileExt,
                   const std::string& outFileExt)
OESimpleAppOptions(const OESystem::OEOptions& opts,
                   const std::string& name,
                   const unsigned inFileType)
OESimpleAppOptions(const OESystem::OEOptions& opts,
                   const std::string& name,
                   const std::string& inFileExt)
OESimpleAppOptions(const OESimpleAppOptions &)
```

Default and copy constructors.

The `opts` represents the sub-options that should be wrapped to extend this options to generate the API for the intended application, where applicable, and the `name` represents the name of the application. The `inFileType` and `outFileType` refers to the input and output file types. The file types can be specified using constants as defined in the *OEFileStringType* namespace, or using a string value. Variation of constructors that does not require an output file can be used when an output is not desired in an application.

GetInFile

```
std::string GetInFile() const
```

See *SetInFile* method.

GetOutFile

```
std::string GetOutFile() const
```

See *SetOutFile* method.

SetInFile

```
bool SetInFile(const std::string value)
```

Sets name of the input file for the application. **Default: None**

SetOutFile

```
bool SetOutFile(const std::string value)
```

Sets name of the output file for the application. **Default: None**

OESubSearch

```
class OESubSearch
```

This class represents *OESubSearch*.

The *OESubSearch* class performs substructure searches, also known as subgraph isomorphism determination, on molecular graphs. In a molecular graph, atoms are considered nodes and bonds are edges. The *OESubSearch* class is capable of performing both colored and uncolored graph comparison. The 'color' properties of a query graph are stored as expression trees associated with atoms and bonds of a query molecule.

See also

- *Substructure Search* section
- *Substructure Search with MDL Queries* chapter

Code Example

- *Perform substructure searches* example

Hint

It is a good practice to call the *OEPrepareSearch* function before calling any of the following methods:

- *OESubSearch.AtomMatch*
- *OESubSearch.Match*
- *OESubSearch.SingleMatch*

If there is any atom or bond property that is not perceived but it is necessary to successfully execute the substructure search, then the above methods throw a warning. These warnings can be suppressed by calling the *OEPrepareSearch* function.

Constructors

```
OESubSearch()
```

Default constructor.

```
OESubSearch(const OESubSearch& rhs)
```

Copy constructor.

```
OESubSearch(const OEQMolBase& qmol)
```

Constructs an *OESubSearch* object using a query molecule (*OEQMolBase*). Query molecules must have atom and bond expressions (see *OEQMolBase.BuildExpressions*) built for the entire molecule before being passed to this constructor. Failure to do so will result in the *OESubSearch* object being constructed as uninitialized.

> **Note**
>
> The *OESubSearch* object will store its own copy of the given query molecule, and the *OESubSearch.Match* method will return matches between copied query and the target.

```
OESubSearch(const OEMolBase& mol, unsigned int aexpr,
            unsigned int bexpr)
```

Constructs an *OESubSearch* object using a molecule and expression options. The atom and bond expression options passed as the second and third arguments to the method are defined in the *OEExprOpts* namespace. The expression options are used to convert the atom and bond data into expression trees which are used during the subgraph search.

```
OESubSearch(const char* smarts, bool allowReorderAtoms = true)
```

Constructs an *OESubSearch* object using a SMARTS pattern that is passed as the first argument to the constructor. The second argument of the method determines the traversal order used in subgraph isomorphism detection. This argument is currently not used, therefore the traversal order always reflects the atom and bond ordering of the passed SMARTS pattern.

operator bool

```
operator bool() const
```

Tests whether initialization of an *OESubSearch* object is successful . If initialization was attempted with an invalid SMARTS pattern or query molecule then this method will return `false`. If initialization was completed successfully this method will return `true`. An *OESubSearch* object is considered to be uninitialized when constructed with the default constructor.

AddConstraint

```
bool AddConstraint(const OEMatchPair<OEAtomBase> &)
bool AddConstraint(const OEMatchPair<OEBondBase> &)
```

The search space of a subgraph isomorphism determination can be restricted by constraining pairs of nodes or edges (atoms or bonds) to be mapped onto one another in all valid subgraph solutions. Failure to satisfy atom or bond pairwise constraints will prevent any subgraph solutions from being identified by any of the match methods in *OESubSearch*. Both *OESubSearch.AddConstraint* methods return `true` if a constraint is added successfully. If the pattern atom or

bond in the *OEMatchPair* does not exist as part of the query molecule created in the initialization of the *OESubSearch* object then *OESubSearch.AddConstraint* will return `false`. Multiple calls *OESubSearch.AddConstraint* using the same pattern atom or bond will cause previously stored constraints to be overwritten as constraints are mutually exclusive. It is impossible to satisfy multiple simultaneous constraints for a single pattern atom or bond, hence the exclusivity.

AtomMatch

```
bool AtomMatch(const OEAtomBase *) const
```

Searches for a single valid subgraph isomorphism starting with the passed target atom mapped to the first pattern atom with which the *OESubSearch* was initialized. If a valid subgraph is identified the method will return `true`. This method is the effective combination of sequential calls to *OESubSearch.AddConstraint* followed by *OESubSearch.SingleMatch*.

ClearConstraints

```
void ClearConstraints()
```

Eliminates all prior match constraints set with the *OESubSearch.AddConstraint* method.

GetMaxMatches

```
unsigned int GetMaxMatches() const
```

Returns the maximum number of subgraph isomorphism solutions that the *OESubSearch* object will identify before terminating the search. A value of zero indicates that no arbitrary limit has been set on the total number of subgraph isomorphism solutions to be identified. By default, the maximum number of matches is set to `1024`.

See also

- *OESubSearch.SetMaxMatches*

GetPattern

```
const OEQMolBase& GetPattern() const
```

Returns the read-only reference to the query molecule contained in an *OESubSearch* object. Const *OEQMolBase* methods can be used to interrogate the returned *OEQMolBase* reference. If the *OESubSearch* object has not been initialized, a reference to an empty molecule will be returned.

Init

```
bool Init(const OEQMolBase &)
```

(Re)initializes an *OESubSearch* object with a query molecule (*OEQMolBase*) reference. The method will return `true` if initialization completes successfully, and `false` upon failure. Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

```
bool Init(const char* smarts, bool allowReorderAtoms = false)
```

(Re)initializes an *OESubSearch* object with a SMARTS pattern, which is parsed to create a query molecule available for substructure search. The method will return `true` if initialization completes successfully, and `false` upon failure. Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly. The second argument of the method determines the traversal order used in subgraph isomorphism detection. This argument is currently not used, therefore the traversal order always reflects the atom and bond ordering of the passed SMARTS pattern.

```
bool Init(const OEMolBase& mol, unsigned int atomexpr,
          unsigned int bondexpr)
```

(Re)initializes an *OESubSearch* object using a molecule and expression options that are used to convert the given molecule into a *OEQMolBase* query object.

mol
The *query* molecule of the maximum common substructure search.

atomexpr
This value has to be any combination of atom expressions of the *OEExprOpts* namespace.

bondexpr
This value has to be any combination of bond expressions of the *OEExprOpts* namespace.

Expression options are used to convert the atom and bond data into expression trees which are used during the substructure search to map atoms and bonds. The method will return `true` if initialization completes successfully, and `false` upon failure.

Note

Prior state information is cleared before initialization, and is lost even if the method fails to initialize properly.

See also

- *OEExprOpts* namespace

Match

```
OESystem::OEIterBase<OEMatchBase>* Match(const OEMolBase& mol,
                                         bool uniquematch = false) const
```

Performs search in order to identify subgraphs of *OEMolBase* which are isomorphic to the graph with which an *OESubSearch* object was initialized. It returns an iterator pointer (*OEIterBase*) over graph matches (*OEMatchBase*), and should be assigned to an *OEIter* < *OEMatchBase* > in order to prevent memory leaks.

mol
The target molecule in which the substructure is identified.

uniquematch
By default all possible subgraphs up to and including the maximum number of matches are returned by these methods. If this parameters is `true` then only the unique matches will be included in the iterator over the matches. A match or subgraph is considered unique if it differs from all other subgraphs found previously by at least one atom. Two subgraph matches which cover the same atoms, albeit in different orders, will be called duplicates and the subgraph found later in the search will be discarded.

SetMaxMatches

```
bool SetMaxMatches(unsigned int m)
```

Sets the maximum number of subgraphs to be determined by the *OESubSearch.Match* method. Once the maximum number of subgraphs have been found the search for additional subgraphs is terminated. By default, maximum number of matches is set to `1024`. The constraint on the maximum number of matches can be removed by calling this method with a value of zero.

See also

- *OESubSearch.GetMaxMatches*

SingleMatch

```
bool SingleMatch(const OEMolBase &) const
```

Returns `true`, if a single subgraph isomorphism is detected in the molecule passed as the function argument. If no subgraph isomorphism is detected the method will return `false`.

See also

- Example program *MolGrep.cs*

OESubSearchDatabase

Attention

This API is currently available in C++ and Python.

class OESubSearchDatabase

The *OESubSearchDatabase* class is designed to perform fast multi-threaded substructure search. Each *OESubSearchDatabase* is associated with one screen type (*OESubSearchScreenTypeBase*) that is set when the database is initialized from a pre-generated binary substructure search file.

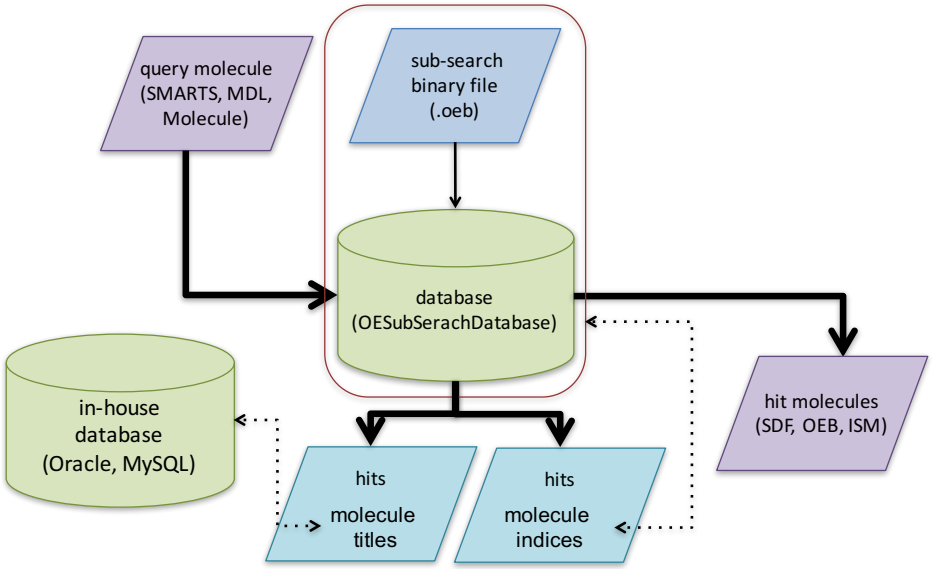


Fig. 6: **Schematic representation of using substructure search database**

The *OESubSearchDatabase* class provides the following search methods:

Method	Returns	Type
NumMatches	number of matched molecules	full search
GetMatchIndices	indices of matched molecules	limited search
GetMatchTitles	titles of matched molecules	limited search
Search	*OESubSearchResult* (indices of molecules, progress of search)	full/limited search (cancelable)

See also

- *OECreateSubSearchDatabaseFile* function to generate database files for *OESubSearchDatabase*.

Code Example

- *Generate database for fast substructure search* example
- *Perform SMARTS search on substructure database file* example

Constructors

```
OESubSearchDatabase(const std::string& dbfile,
                    const unsigned memtype = OESubSearchDatabaseType::Default,
                    const unsigned numprocesssors = 0u)
OESubSearchDatabase(const unsigned memtype = OESubSearchDatabaseType::Default,
                    const unsigned numprocesssors = 0u)
```

Constructs an *OESubSearchDatabase* object.

dbfile
The name of the file that contains the substructure search data (molecules with pre-generated screens). The file has to be generated with the *OECreateSubSearchDatabaseFile* function. If no file name is defined, the database has to be initialized by calling the *OESubSearchDatabase.Open* method.

memtype
Defines the type of the substructure database. This value has to be from the *OESubSearchDatabaseType* namespace.

numprocesssors
The maximum number of threads that can be used to search the database. When set to `0`, the number of processors used will be the number returned by the *OEGetNumProcessors* function. If `numprocesssors == 1`, all searches are executed using the main thread only.

GetDatabaseType

```
unsigned GetDatabaseType() const
```

Returns the type of the substructure search database. The return value is taken from the *OESubSearchDatabaseType* namespace.

GetMatchIndices

```
OESystem::OEIterBase<size_t>* GetMatchIndices(const OESubSearchQuery &) const
```

Returns the indices of molecules matched to the given query (*OESubSearchQuery*). The search terminates if the limit set for the query is reached (*GetMaxMatches*). The matched molecules can be retried from the database by using the *OESubSearchDatabase.GetMolecule* method.

GetMatchTitles

```
OESystem::OEIterBase<std::string>* GetMatchTitles(const OESubSearchQuery &) const
```

Returns the titles of the molecules matched to the given query (*OESubSearchQuery*). The search terminates if the limit set for the query is reached (*GetMaxMatches*).

GetMolecule

```
bool GetMolecule(OEMolBase& mol, const size_t index) const
```

Retrieves the molecule stored in the database with the specified index.

See also

- *OESubSearchDatabase.GetTitle* method

GetScreenType

```
const OESubSearchScreenTypeBase* GetScreenType() const
```

Returns the screen type of the *OESubSearchDatabase* object. An *OESubSearchDatabase* object can only store screens with identical types.

GetTitle

```
std::string GetTitle(const size_t index) const
```

Retrieves the title of the molecule stored in the database with the specified index.

See also

- *OESubSearchDatabase.GetMolecule* method

IsValid

```
bool IsValid() const
```

Returns whether the *OESubSearchDatabase* object is valid.

NumMatches

```
size_t NumMatches(const OEQMolBase& query) const
```

Returns the number of molecules matched to the given query (*OESubSearchQuery*) in the entire database.

NumMolecules

```
size_t NumMolecules() const
```

Returns the number of molecules stored in the *OESubSearchDatabase* object.

NumProcessors

```
unsigned NumProcessors() const
```

Returns the maximum number of threads that can be used to search the database. This is determined when the database is initialized.

Open

```
bool Open(oemolistream& ifs,
          OESystem::OETracerBase& tracer = OESystem::OENoTracer)
bool Open(const std::string & filename,
          OESystem::OETracerBase& tracer = OESystem::OENoTracer)
```

Initializes the substructure search database. These methods return `true` if the database was successfully initialized.

ifs
The molecule stream from which the substructure search database is initialized.

filename
The name of the file from which the substructure search database is initialized.

tracer
The tracer that can be used to report the progress of the database load.

See also

- *OEConsoleProgressTracer* class

Search

```
unsigned Search(OESubSearchResult& result,
                const OESubSearchQuery& query) const
```

This search method is designed for interactive searches.

result
The *OESubSearchResult* object that stores the result of the search along with the progress of the search. The *OESubSearchResult* is updated regularly during the search as more and more matches identified. It also provides a mechanism to cancel a long running search via *OESubSearchResult.Cancel*.

query
The *OESubSearchQuery* object that stores the query molecule with search parameters.

The *OESubSearchDatabase.Search* method returns:

- *OESubSearchStatus.Uninitialized* if the search can not be executed
- *OESubSearchStatus.Finished*, if the search successfully finished
- *OESubSearchStatus.Canceled*, if the search was interrupted

See also

- *OESubSearchStatus* namespace

OESubSearchQuery

Attention

This API is currently available in C++ and Python.

```
class OESubSearchQuery
```

The *OESubSearchQuery* class is used to submit queries to be searched in a database (*OESubSearchDatabase*).

See also

- *OESubSearchDatabase.Search* method
- *OESubSearchDatabase.GetMatchIndices* method
- *OESubSearchDatabase.GetMatchTitles* method

Code Example

- *Perform SMARTS search on substructure database file* example

Constructors

```
OESubSearchQuery(const OEQMolBase& query,
                 const size_t maxmatches = 1000u)
```

Creates an *OESubSearchQuery* object.

query
The query molecule (*OEQMolBase*).

maxmatches
The maximum number of matches that will be kept.

SetFilter

```
void SetFilter(const OESystem::OEUnaryPredicate<OEMolBase>&)
```

Sets a molecule predicate that can be used to filter out molecules based on molecular properties other than the existence of a certain substructure.

Note

During the search, the predicate set by the *OESubSearchQuery.SetFilter* method is utilized after the screening phase and before the atom-by-atom validation of the substructure search match. Note that to minimize the memory footprints, the *OESubSearchDatabase* only stores the molecular graphs (no coordinates) and the titles of the molecules.

See also

- *Predicate Functors* chapter

GetMaxMatches

```
size_t GetMaxMatches() const
```

Returns the maximum match limit when searching *OESubSearchDatabase*. The *OESubSearchDatabase.GetMatchIndices* and *OESubSearchDatabase.GetMatchTitles* methods will terminate when this limit is reached. The default is `1000`.

SetMaxMatches

```
void SetMaxMatches(const size_t limit)
```

Sets the maximum match limit.

Note

While there is no upper limit on how many matches can be retrieve by the search, it is not recommended to set this limit very high (>10K). Searching a very large database with a very generic query can result in internally storing millions of indices or titles. The total number of matches can be determined (without storing all matches) either using the *OESubSearchDatabase.NumMatches* method or via the *OESubSearchResult.NumTotalMatches* counter when using the *OESubSearchDatabase.Search* method.

OESubSearchResult

Attention

This API is currently available in C++ and Python.

```
class OESubSearchResult
```

The *OESubSearchResult* class is used to store the results of a substructure search along with reporting the progress of the search (*OESubSearchDatabase.Search*). The *OESubSearchResult* class provides progress with updating the following counters:

Counter	Description
NumTargets	number of molecules in the database
NumScreened	number of molecules eliminated by their screen
NumSearched	number of molecules searched
NumTotalMatches	total number of matches identified
NumMatches	total number of matches kept

If the search is not canceled at the end of the search:

$$number\ of\ targets = number\ of\ screened + number\ of\ searched$$

The success of the screen (in percentages) is defined as:

$$\frac{number\ of\ screened + number\ of\ total\ matches}{number\ of\ targets} * 100.0$$

Where `100.0%` means that no false positive was detected i.e. each molecule that passed the screening phase was a real match.

See also

- *OESubSearchDatabase.Search* method

Code Example

- *Perform SMARTS search on substructure database file* example

Constructors

```
OESubSearchResult()
```

Default constructor that creates an empty result object.

Cancel

```
void Cancel()
```

Cancels the ongoing substructure search. The search might not be immediately terminated, but no new molecules will be submitted for substructure search. The status of the search will be *OESubSearchStatus.Canceled*.

GetMatchIndices

```
OESystem::OEIterBase<size_t>* GetMatchIndices() const
```

Returns the indices of the matched molecules. The matched molecules can be retried from the database by using the *OESubSearchDatabase.GetMolecule* method.

GetSearchStatus

```
unsigned GetSearchStatus() const
```

Returns the status of the substructure search defined in the *OESubSearchStatus* namespace.

IsValid

```
bool IsValid() const
```

Returns whether the *OESubSearchResult* object is valid.

NumMatches

```
size_t NumMatches() const
```

Returns the number of kept matches which can be retrieved via the *OESubSearchResult.GetMatchIndices* method. After the search limit is reached (defined by *GetMaxMatches*), only the *NumTotalMatches* counter will be updated.

See also

- *OESubSearchResult.NumTotalMatches* method

NumScreened

```
size_t NumScreened() const
```

Returns the number of molecules that could be eliminated by their screens.

See also

- *OEScreen* function

NumSearched

```
size_t NumSearched() const
```

Returns the number of molecules that have to be searched i.e. the number of times *OESubSearch* has to be used.

NumTargets

```
size_t NumTargets() const
```

Returns the number of target molecules i.e the number of molecules in the database.

NumTotalMatches

```
size_t NumTotalMatches() const
```

Returns the total number of matches in the database.

See also

- *OESubSearchResult.NumMatches* method

OESubSearchScreen

Attention

This API is currently available in C++ and Python.

```
class OESubSearchScreen : public OESystem::OEBitVector
```

OESubSearchScreen class represents substructure search screens that encode local and global features of query and target molecules. These screens can be used to accelerate the substructure search since target molecules that clearly can not be matched to the query can be rapidly eliminated. Only molecules that pass the screening phase are subjected to the more expensive atom-by-atom validation when using the search methods of the *OESubSearchDatabase* class. An *OESubSearchScreen* objects are typed bit-vectors (*OEBitVector*). The type determines what kind of queries in mind the screen was designed for (SMARTS, MDL query, or molecule query).

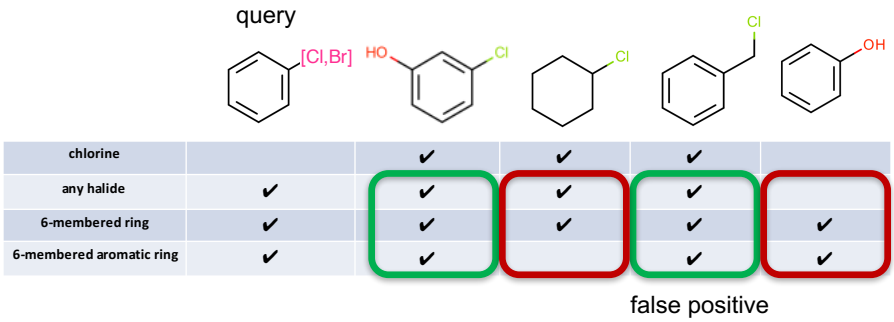


Fig. 7: **Example of using substructure search screens to eliminate target molecules**

See also

- *OESubSearchScreenTypeBase* class
- *OESubSearchScreenType* namespace
- *OEMakeSubSearchQueryScreen* and *OEMakeSubSearchTargetScreen* functions

Constructors

```
OESubSearchScreen()
```

The default constructor creates an uninitialized *OESubSearchScreen* object. The object can be initialized by calling either the *OEMakeSubSearchQueryScreen* function for query molecules or the *OEMakeSubSearchTargetScreen* function for target molecules.

```
OESubSearchScreen(const OESubSearchScreen &)
```

Copy constructor.

GetScreenType

```
const OESubSearchScreenTypeBase* GetScreenType() const
```

Returns the type of the screen (*OESubSearchScreenTypeBase*).

IsValid

```
bool IsValid() const
```

Returns whether the *OESubSearchScreen* object has a valid screen type.

OESubSearchScreenTypeBase

Attention

This API is currently available in C++ and Python.

```
class OESubSearchScreenTypeBase
```

The *OESubSearchScreenTypeBase* class provides an interface to type *OESubSearchScreen* objects. The type determines what kind of queries in mind the screen was designed for

- SMARTS patterns
- *MDL* queries
- simple molecule queries with no complex query features

Screen types can not be constructed by themselves. The type of an *OESubSearchScreen* object is determined when it is initialized by calling either the *OEMakeSubSearchQueryScreen* or the *OEMakeSubSearchTargetScreen* function.

Each screen type has a version number. Version numbers are introduced in order to keep track of changes in the screen generation algorithm itself.

See also

- *OESubSearchScreenType* namespace

GetName

```
std::string GetName() const
```

Returns the name of the screen type.

GetScreenType

```
unsigned int GetScreenType() const
```

Returns the screen type defined in the *OESubSearchScreenType* namespace.

GetScreenTypeString

```
std::string GetScreenTypeString() const
```

Returns the name and the version of the screen type in the string format.

GetSize

```
unsigned int GetSize() const
```

Returns the number of bits in of the screen type.

GetVersion

```
unsigned short GetVersion() const
```

Returns the current version of the screen type.

GetVersionString

```
std::string GetVersionString() const
```

Returns the current version of the screen type in the string format, such as `0.9.0` .

> **Note**
>
> The version number of various screen types will not be changed with each release. It will be incremented only if modifications or bug fixes to the corresponding algorithm would result in generating a different screen for the same molecule.
>
> Screens with an old version number will **not** be supported. But rather it will be required to regenerate the screen databases.

OETokenizerBase

```
class OETokenizerBase
```

This is an abstract base class used to represent an object that can efficiently split up a molecular file into a stream of tokens, each token representing one molecule in the stream. Implementations of this class are returned by the *OEGetTokenizer* function. This class is expected to do the minimal amount of string parsing possible to determine the next chunk of bytes representing a molecule.

GetNextToken

```
std::string* GetNextToken(OEPlatform::oeistream& ifs)=0
```

Returns a chunk of bytes representing the next molecule in the stream `ifs`. The stream of bytes is expected to be in the same file format as was passed to the *OEGetTokenizer* factory function used to construct this object.

ParseTitle

```
std::string ParseTitle(const std::string& data) const = 0
```

Return the string that *OEMolBase.GetTitle* would return for the molecule record data passed in. The implementations of this method are designed to do the minimal amount of parsing required to retrieve the title.

OETorsion

```
struct OETorsion
```

An object to contain the information necessary to represent a torsion: a 3D rotation around the vector defined by the coordinates of `a2` and `a3` and an angle defined by `a1` and `a4`. The angle is stored in radians instead of degrees

See also

- *OEGetTorsions*
- *OEConfBase.SetTorsion*
- *OEMCMolBase.NewConf*
- OESzybki.SetTorsionConstraint

Constructors

```
OETorsion()
OETorsion(OEAtomBase* a1, OEAtomBase* a2, OEAtomBase* a3,
          OEAtomBase* a4, double rads)
```

Construct a torsion object containing 4 atoms and an angle in radians.

OETrans

```
class OETrans
```

This class represents *OETrans*.

The *OETrans* class is a container of high-level molecular transformations. It can be used to hold and wrap a series of molecular transformations, which can be applied to *OEConfBase* objects with a single function call. The *OETrans* container keeps its own copies of the *OETransBases*.

Constructors

```
OETrans()
```

Default constructor.

```
OETrans(const OETrans& rhs)
```

Copy constructor.

```
OETrans(const OETransBase* t)
OETrans(const OETransBase& tr)
```

Creates a *OETrans* transformation container and adds a copy of the *OETransBase* object to the container.

operator bool

```
operator bool() const
```

Returns `true`, if at least one transformation (*OETransBase*) is stored in the *OETrans* object.

Clear

```
void Clear()
```

Removes all transformations (*OETransBase*) from the *OETrans* object.

Compress

```
void Compress()
```

Extract

```
template<class T, class U>
void Extract(T* rmat, U* trans, bool clobber = true) const
```

Extract the *OETransBase* transformation as a rotation matrix, `rmat`, and a translation vector, `trans`.

GetInverse

```
bool GetInverse(OETrans& inverse)
```

GetTransforms

```
OESystem::OEIterBase<OETransBase>* GetTransforms() const
```

Returns an iterator over the transformations (*OETransBase*) stored in the *OETrans* object.

PushBack

```
void PushBack(const OETransBase& t)
```

Makes a copy of the *OETransBase* transformation and adds it to the end of the *OETrans* container.

PushFront

```
void PushFront(const OETransBase& t)
```

Makes a copy of the *OETransBase* transformation and adds it to the front of the *OETrans* container.

Transform

```
void Transform(OEMolBase* mol) const
void Transform(OEConfBase* conf) const
```

Applies the transformation specified by the *OETransBase* object to the molecule or conformer passed.

```
void Transform(float* coords, unsigned int ncoords,
               unsigned int dim) const
void Transform(double* coords, unsigned int ncoords,
               unsigned int dim) const
```

Applies all transformations stored in the *OETrans* object to the a set of coordinates ('coords'). The 'row' argument specifies the number of coordinates in 'coords' array, while the 'dim' argument specifies the dimension of the coordinates. These methods presumes that the size of 'coords' at least $rows * dim$.

OETransBase

```
class OETransBase
```

The *OETransBase* is the abstract base class which describes the interface for all of the high-level molecular transformation objects. It allows generic application of many manipulations (such as *translation*, *Euler rotation*, *rotation by matrix*, *quaternion rotation*) to conformers. All of the default transformation currently provided carry out rigid transformations of molecule positions rather than conformational modifications. These are provided in the *OEMath* namespace described below.

The following classes derive from this class:

- *OEEuler*
- *OEQuaternion*
- *OERotMatrix*
- *OETranslation*

CreateCopy

```
OETransBase* CreateCopy() const = 0
```

It is a virtual const constructor which allows copying of concrete derived objects using a reference to this base class.

Extract

```
void Extract(float *, float *) const = 0
void Extract(double *, float *) const = 0
void Extract(float *, double *) const = 0
void Extract(double *, double *) const = 0
```

GetInverse

```
OETransBase* GetInverse() const = 0
```

Transform

```
void Transform(OEMolBase* mol) const
void Transform(OEConfBase* conf) const
```

Applies the transformation specified by the *OETransBase* object to the molecule or conformer passed.

```
void Transform(float* coords, unsigned int rows, unsigned int dim) const
void Transform(double* coords, unsigned int rows, unsigned int dim) const
```

Applies the transformation specified by the *OETransBase* object to the a set of coordinates ('coords'). The 'row' argument specifies the number of coordinates in 'coords' array, while the 'dim' argument specifies the dimension of the coordinates. These methods presumes that the size of 'coords' at least $rows * dim$. Both methods modify the coordinates in place.

OETranslation

```
class OETranslation : public OETransBase
```

This class represents *OETranslation*.

The *OETranslation* class performs Cartesian translation.

The following methods are publicly inherited from *OETransBase*:

CreateCopy	*GetInverse*
Extract	*Transform*

Constructors

```
OETranslation()
OETranslation(const float* vec)
OETranslation(const double* vec)
OETranslation(float dx, float dy, float dz)
OETranslation(double dx, double dy, double dz)
```

Default and copy constructors.

CreateCopy

```
OETransBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OETranslation* object is dynamically allocated and owned by the caller.

Extract

```
void Extract(float* r, float* t) const
void Extract(double* r, float* t) const
void Extract(float* r, double* t) const
void Extract(double* r, double* t) const
```

GetInverse

```
OETransBase* GetInverse() const
```

GetTranslation

```
void GetTranslation(float* vec) const
void GetTranslation(double* vec) const
```

OEUnaryToBinaryAnd

```
template<typename T>
class OEUnaryToBinaryAnd : public OESystem::OEBinaryPredicate<T, T>
```

This class represents *OEUnaryToBinaryAnd*.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEUnaryToBinaryAnd(const OEUnaryToBinaryAnd& rhs)
OEUnaryToBinaryAnd(const OESystem::OEUnaryPredicate<T>& p)
```

Default and copy constructors.

operator()

```
bool operator()(const T& arg1, const T& arg2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<T, T, bool>* CreateCopy() const
```

OEUnaryToBinaryOr

```
template<typename T>
class OEUnaryToBinaryOr : public OESystem::OEBinaryPredicate<T, T>
```

This class represents *OEUnaryToBinaryOr*.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEUnaryToBinaryOr(const OEUnaryToBinaryOr& rhs)
OEUnaryToBinaryOr(const OESystem::OEUnaryPredicate<T>& p)
```

Default and copy constructors.

operator()

```
bool operator()(const T& arg1, const T& arg2)
bool operator()(const T& arg1, const T& arg2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<T, T, bool>* CreateCopy() const
```

OEUnAtomPredParameter

```
class OEUnAtomPredParameter : public OEPredicateParameter<OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase>>
```

The *OEUnAtomPredParameter* represents parameter that has value of *OEUnaryPredicate* type.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*

- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OEPredicateParameter*:

- *GetDefault* and *SetDefault*
- *GetSetting*
- *GetValue* and *SetValue*

Constructors

```
OEUnAtomPredParameter()
OEUnAtomPredParameter(const std::string& name)
OEUnAtomPredParameter(const OEUnAtomPredParameter&)
```

Default and copy constructors.

Constructs an *OEUnAtomPredParameter* instance using the specified set of parameters.

OEUnBondPredParameter

```
class OEUnBondPredParameter : public OEPredicateParameter<OESystem::OEUnaryPredicate
↪<OEChem::OEBondBase>>
```

The *OEUnBondPredParameter* represents parameter that has value of *OEUnaryPredicate* type.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*

- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OEPredicateParameter*:

- *GetDefault* and *SetDefault*
- *GetSetting*
- *GetValue* and *SetValue*

Constructors

```
OEUnBondPredParameter()
OEUnBondPredParameter(const std::string& name)
OEUnBondPredParameter(const OEUnBondPredParameter&)
```

Default and copy constructors.

Constructs an *OEUnBondPredParameter* instance using the specified set of parameters.

OEUniMolecularRxn

```
class OEUniMolecularRxn
```

The *OEUniMolecularRxn* class is designed to apply a chemical transformation to a molecule. The primary purpose of the class is to provide an efficient means of applying normalization reactions, although there are few restrictions on the types of chemical transformations that can be applied. The reaction used to initialize the class must contain only a single reactant. The provided transformation is applied iteratively until no further reactant patterns are matched. Caution must be used in defining the transformation as it is possible to provide a reaction that will cause a *OEUniMolecularRxn* object to iterate until memory is exhausted. For example, a reaction for which the reactant pattern matches the transformed product molecule will reapply the transformation indefinitely and must be avoided. Note, however, that atoms and bonds created by a transformation are excluded from involvement in iterative applications of the transformation.

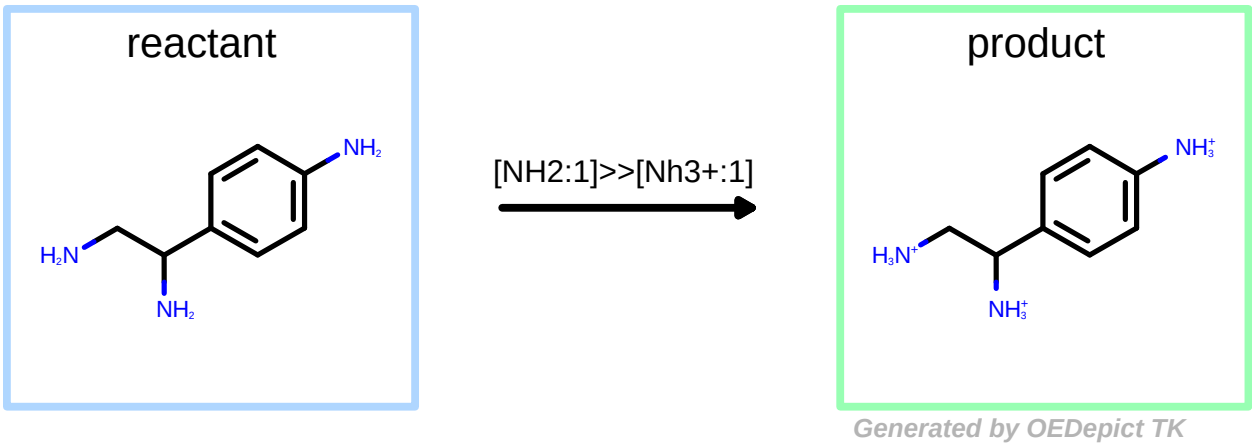


Fig. 8: **OEUniMolecularRxn**
Example of amine nitrogen protonation using 'OEUniMolecularRxn'. Transformations are applied to the starting material recursively

See also

- *Reactions* chapter
- SMIRKS
- *OELibraryGen* class
- Example program *UniMolRxn.cs*

Constructors

```
OEUniMolecularRxn()
```

Default constructor.

```
OEUniMolecularRxn(const OEUniMolecularRxn& rhs)
```

Copy constructor.

```
OEUniMolecularRxn(const char * smirks, bool strictSmirks)
```

Creates a *OEUniMolecularRxn* object with a (SMIRKS) pattern. Attempts to initialize an *OEUniMolecularRxn* object with a reaction that has more than one reactant will result in an initialization failure. Initialization success can be tested using the operator *OEUniMolecularRxn.operator bool* method. The second argument to the constructor is used to specify the interpretation of the SMIRKS semantics. By default, the `smirks` string is interpreted using strict semantics via a value `true`.

Passing a `smirks` value of `false` to the second method argument will relax the strict SMIRKS restrictions. A mapped reactant atom that does not have a corresponding mapped product atom is valid, and will be destroyed as part of the reaction. Unmapped reactant atoms will be used to match the reactant pattern, but are not destroyed when the reaction is applied. The resulting SMIRKS like reactions may therefore be more easily readable by humans as fewer atoms may be required to be mapped.

See also

- *OEUniMolecularRxn.Init* method
- SMIRKS definition describing the strict SMIRKS semantics

```
OEUniMolecularRxn(const char * smirks,
                  const OEChem::OEUniMolecularRxnOptions& options =␣
↪OEUniMolecularRxnOptions())
```

Creates a *OEUniMolecularRxn* object with a (SMIRKS) pattern. The options class argument will be interrogated to determine the desired strict `smirks` interpretation requested for the (SMIRKS) processing.

See also

- *OEUniMolecularRxnOptions.GetStrictSmirks* method
- *OEUniMolecularRxn.Init* method
- SMIRKS definition describing the strict SMIRKS semantics

operator()

```
bool operator()(OEMolBase& mol) const
```

Applies the transformation that the *OEUniMolecularRxn* object contains on the passed *OEMolBase* object. Returns `true` if the reactant pattern matches at least one time in the passed molecule.

If the *OEUniMolecularRxn* fails to apply any transformations because the reactant pattern failed to match the molecule, the method will return `false`.

operator bool

```
operator bool() const
```

Returns `true`, if the *OEUniMolecularRxn* object was initialized with no failures using either the constructors or one of the *OEUniMolecularRxn.Init* methods.

The method will return `false` when the *OEUniMolecularRxn* object has not been properly initialized as it will be unable to apply transform operations to molecules.

GetClearCoordinates

```
bool GetClearCoordinates() const
```

Returns whether the coordinates of a molecule are cleared before performing the chemical transformation.

See also

- *OEUniMolecularRxn.SetClearCoordinates* method

GetOptions

```
OEUniMolecularRxnOptions& GetOptions()
const OEUniMolecularRxnOptions& GetOptions() const
```

Returns the internal options class in the *OEUniMolecularRxn* instance. A non-const instance supports changing the option control values for options that are active during transformation application. Note that the `strict` flag processing setting occurs when the SMIRKS or *OEQMolBase* transformation is processed and not during application of the transformation.

See also

- *OEUniMolecularRxnOptions* class

GetValidateKekule

```
bool GetValidateKekule() const
```

Returns whether the transformation validates the Kekulé form of the returned products.

See also

- *OEUniMolecularRxn.SetValidateKekule* method
- *Product Kekulization* section

Init

```
bool Init(const char* smirks,
          const OEUniMolecularRxnOptions& options = OEUniMolecularRxnOptions())
bool Init(const OEQMolBase& qmol,
          const OEUniMolecularRxnOptions& options = OEUniMolecularRxnOptions())
```

The first two method overloads initialize an *OEUniMolecularRxn* object with a `smirks` pattern and either the explicitly provided `strict` flag setting or via the `strict` setting from the provided options class.

The next two method overloads initialize an *OEUniMolecularRxn* object with a *OEQMolBase* object and either the explicitly provided `strict` flag setting or via the `strict` setting from the provided options class.

See also

- *OEUniMolecularRxnOptions.GetStrictSmirks* method

```
bool Init(const char* smirks, bool strict)
bool Init(const OEQMolBase &, bool strictSmirks = true)
```

Initializes an *OEUniMolecularRxn* object with a query molecule (*OEQMolBase*). If query molecule is created using the *OEParseSmirks* function, and if the reaction is desired to be interpreted using strict SMIRKS semantics, then the default `true` value for the second argument to the method should be used.

Passing a value of `false` to the second argument will relax the restrictions imposed by strict SMIRKS semantics.

See also

- SMIRKS definition describing the strict SMIRKS semantics

SetClearCoordinates

```
bool SetClearCoordinates(bool clearcoords)
```

Sets whether the coordinates of a molecule are cleared before performing the chemical transformation.

Note

The *OEUniMolecularRxn* modifies the molecular graph when applying the transformations. Generating valid 2D or 3D coordinates for the products is beyond the scope of the *OEUniMolecularRxn* class.

It is recommended to either clear the coordinates by invoking the *OEUniMolecularRxn.SetClearCoordinates* method or regenerate the coordinates of the molecule afterwards.

If no transformation is applied the molecule will not be changed.

See also

- *OEUniMolecularRxn.GetClearCoordinates* method

SetValidateKekule

```
bool SetValidateKekule(bool validate)
```

Sets whether the generated products are simply returned after a Kekulization attempt, or if the Kekulé form is validated more rigorously. If the extra validation is enabled and an invalid Kekulé form is identified, alternatives are tried by adding and removing implicit hydrogens from aromatic nitrogens to produce a valid Kekulé form. No further attempt is made to return the most reasonable or chemically sensible result. By default, the extra Kekulé validation is enabled.

See also

- *Product Kekulization* section

OEUniMolecularRxnOptions

```
class OEUniMolecularRxnOptions
```

This class represents the *OEUniMolecularRxnOptions* options class that encapsulates properties to control transforms using *OEUniMolecularRxn* and *OEGetUniMolecularRxnIter*.

See also

- *OEUniMolecularRxn* class

- *OEGetUniMolecularRxnIter*

Constructors

```
OEUniMolecularRxnOptions()
```

Default constructor that initializes an *OEUniMolecularRxnOptions* object with the following properties:

Table 2: **Default parameters of OEUniMolecularRxnOptions**

Option	Default value
Strict SMIRKS	`true`
Validate Kekule	`true`
Clear Coordinates	`false`
Hydrogen Conversions	`true`
Fix Valences	*OEUniMolecularRxnFixValence.Explicit*
Altered Tag	empty
Max Matches	10000

```
OEUniMolecularRxnOptions(const OEUniMolecularRxnOptions& rhs)
```

Copy constructor.

GetAlteredTag

```
std::string GetAlteredTag() const
```

Return the desired generic data tag that captures the boolean state indicating specific atoms that are modified by the transformation. If empty or blank, no generic data is returned on the transformation product atoms.

See also

- *OEUniMolecularRxnOptions.SetAlteredTag* method

GetClearCoordinates

```
bool GetClearCoordinates() const
```

Returns whether the coordinates of a molecule are cleared before performing the chemical transformation.

See also

- *OEUniMolecularRxnOptions.SetClearCoordinates* method

GetFixValence

```
unsigned GetFixValence() const
```

Returns what type of valence corrections should be applied to the transformation products. The legal values are taken from *OEUniMolecularRxnFixValence*.

See also

- *OEUniMolecularRxnOptions.SetFixValence* method

GetHydrogenConversions

```
bool GetHydrogenConversions() const
```

Returns whether hydrogen count conversions are allowed to be performed on transformation targets based on the transformation SMIRKS requirements.

See also

- *OEUniMolecularRxnOptions.SetHydrogenConversions* method
- *OEPrepareSearch* function

GetMaxMatches

```
unsigned GetMaxMatches() const
```

The maximum number of transformations returned or applied. A value of `0` indicates there is no constraint. Note that if this value is low, the transformations returned will be arbitrarily truncated to this limit.

See also

- *OEUniMolecularRxnOptions.SetMaxMatches* method

GetStrictSmirks

```
bool GetStrictSmirks() const
```

Returns the current setting of the `strict` flag that controls how to interpret the SMIRKS for the transformation. This setting value controls the interpretation of the SMIRKS semantics. By default, the SMIRKS string is interpreted using strict semantics (i.e., with the setting value of `true`).

See also

- *OEUniMolecularRxnOptions.SetStrictSmirks* method

GetValidateKekule

```
bool GetValidateKekule() const
```

Returns whether the transformation validates the Kekulé form of the returned products.

See also

- *OEUniMolecularRxnOptions.SetValidateKekule* method

SetAlteredTag

```
bool SetAlteredTag(const std::string& alteredtag)
```

Sets the desired generic data tag value used to capture boolean data indicating specific atoms that are modified by the transformation. If empty or blank, no generic data is returned on the transformation product atoms.

See also

- *OEUniMolecularRxnOptions.GetAlteredTag* method

SetClearCoordinates

```
bool SetClearCoordinates(bool clearcoords)
```

Sets whether the coordinates of a molecule should be cleared before performing the chemical transformation.

Note

The *OEUniMolecularRxn* modifies the molecular graph when applying the transformations. Generating valid 2D or 3D coordinates for the products is beyond the scope of the *OEUniMolecularRxn* class.

It is recommended to either clear the coordinates by invoking the *OEUniMolecularRxn.SetClearCoordinates* method or regenerate the coordinates of the molecule afterwards.

If no transformation is applied the molecule will not be changed.

See also

- *OEUniMolecularRxnOptions.GetClearCoordinates* method

SetFixValence

```
bool SetFixValence(unsigned fixvalence)
```

Sets the type of valence corrections that should be applied to the transformation products. The legal values are taken from *OEUniMolecularRxnFixValence*.

See also

- *OEUniMolecularRxnOptions.GetFixValence* method

SetHydrogenConversions

```
bool SetHydrogenConversions(bool allowed)
```

Sets whether hydrogen count conversions are allowed to be performed on transformation targets based on the transformation SMIRKS requirements. If `true`, *OEPrepareSearch* is called to reconcile the hydrogen state between the target and query as well as any other perception information needed for the query (aromaticity, etc.).

See also

- *OEUniMolecularRxnOptions.GetHydrogenConversions* method
- *OEPrepareSearch* function

SetMaxMatches

```
bool SetMaxMatches(unsigned maxmatches)
```

Sets the maximum number of transformations that should be returned or applied. Setting the value to `0` removes the constraint. Note that if this value is low, the transformations returned will be arbitrarily truncated to this limit.

See also

- *OEUniMolecularRxnOptions.GetMaxMatches* method

SetStrictSmirks

```
bool SetStrictSmirks(bool strict)
```

Sets the desired value of the `strict` flag that controls how to interpret the SMIRKS for the transformation. This setting value controls the interpretation of the SMIRKS semantics. By default, the SMIRKS string is interpreted using strict semantics (i.e., with the setting value of `true`).

See also

- *GetStrictSmirks* method
- SMIRKS definition describing the strict SMIRKS semantics

SetValidateKekule

```
bool SetValidateKekule(bool validatekekule)
```

Sets whether the generated products are simply returned after a Kekulization attempt, or if the Kekulé form is validated more rigorously. If the extra validation is enabled and an invalid Kekulé form is identified, alternatives are tried by adding and removing implicit hydrogens from aromatic nitrogens to produce a valid Kekulé form. No further attempt is made to return the most reasonable or chemically sensible result. By default, the extra Kekulé validation is enabled.

See also

- *OEUniMolecularRxnOptions.GetValidateKekule* method

OEVectorBindings

```
class OEVectorBindings
```

This class represents *OEVectorBindings*.

The *OEVectorBindings* class is used to store a set of vector bindings or lexigraphic replacements that may appear in a SMARTS pattern.

A vector binding is a SMARTS pattern bound to a name. For example, the name `[$HALO]` can be used to represent the pattern `[$(F,Cl,Br,I)]` in order to make a pattern more human readable. The *OEVectorBindings* class converts SMARTS patterns written using vector bindings (human readable form) to the corresponding machine readable form by performing a name to pattern replacement.

Constructors

```
OEVectorBindings()
OEVectorBindings(const OEVectorBindings &)
```

Default and copy constructors.

Add

```
bool Add(const char* label, const char* pattern)
```

Adds one vector binding to an *OEVectorBindings* object. The 'label' given as the first method argument is the name that appears in the human readable version of a SMARTS, while the 'pattern' given as the second method argument is the SMARTS that should be used to replace the ''label". If the 'pattern' is parsed correctly and the association is made in the *OEVectorBindings* object, the method will return `true`. If the 'pattern' is invalid or the association cannot be made the method will return `false`. All attempts to associate a SMARTS pattern with a particular 'label' after the first association succeeds will result subsequent failures to create new associations.

Get

```
bool Get(const OEExprBase *&expr, const char *&label,
         const char* smarts) const
```

Retrieves the 'label' and corresponding *OEExprBase* pointer reference as the first and second method arguments, respectively, given a pointer to a position within a *smarts* string given as the final argument. If the 'smarts' string position

points to the beginning of a vector bound 'label' contained in the *OEVectorBindings* object then the method will return `true`. If the vector binding cannot be identified then the method will return `false`.

4.1.2 OEChem Constants

OEChem

This namespace contains the following constants:

OEAroModelDaylight

Deprecated See *Daylight* aromaticity model.

See also

- *Aromaticity Models in OEChem TK* section
- *OEAssignAromaticFlags* function

OEAroModelOpenEye

Deprecated See *OpenEye* aromaticity model.

See also

- *Aromaticity Models in OEChem TK* section
- *OEAssignAromaticFlags* function

OEAroModelTripos

Deprecated See *Tripos* aromaticity model.

See also

- *Aromaticity Models in OEChem TK* section
- *OEAssignAromaticFlags* function

OEAroModelMMFF

Deprecated See *MMFF* aromaticity model.

See also

- *Aromaticity Models in OEChem TK* section
- `OEAssignAromaticFlags` function

OEAroModelMDL

Deprecated See `MDL` aromaticity model.

See also

- *Aromaticity Models in OEChem TK* section
- `OEAssignAromaticFlags` function

FzFalse

This is a globally available *OEFuzzy* object that represents fuzzy false value.

FzTrue

This is a globally available *OEFuzzy* object that represents fuzzy true value.

FzMaybe

This is a globally available *OEFuzzy* object that represents fuzzy maybe value.

Preliminary OEChem Constants

OEAminoAcidType

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The `OEAminoAcidType` namespace encodes symbolic constants used as bit-masks to categorize amino-acids.

A monomer such as *N-alpha-methyl-L-Alanine* (defined with `[H:1]N(C)[C@@H](C)C(=O)[OH:2]`) belongs to both `OEAminoAcidType.Alpha` and `OEAminoAcidType.AlphaMethylated` categories.

The namespace contains the following constants:

Alpha

Standard amino acid where the amino group attached to the alpha-carbon.

AlphaMethylated

Amino acid where a methyl group is attached to alpha-carbon of its backbone.

Beta

Amino amino with amino group attached to the beta-carbon.

BetaMethylated

Amino acid where a methyl group is attached to beta-carbon of its backbone.

Delta

Standard amino acid where the amino group attached to the delta-carbon.

Gamma

Standard amino acid where the amino group attached to the gamma-carbon.

NMethylated

Amino acid where a methyl group is attached to amino group of its backbone.

NotAminoAcid

Indicates monomers that are not amino-acids.

Undefined

See also

- *OEGetAminoAcidType* function
- *OEHasAminoAcidType* functor
- *OEAminoAcidTypeToString* function

OEHelmGenerationReturnCode

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The *OEHelmGenerationReturnCode* namespace contains codes representing the success or various issues of HELM string generation when using the *OEMolToHelm* function.

See also

- *OEHelmGenerationResult* class

Success

Represents that the HELM is **successfully** generated.

InvalidCodeSet

Returned if the the monomers set used for HELM generation does not have the specified code-set.

InvalidMonomerGraph

Returns if after the monomers mapped to the molecule, the monomer graph can not be fully traversed to encode all the monomer components in the HELM string.

InvalidUnmappedFragment

Returned if there is any unmapped monomer in the molecule that can not be represented in the HELM as an embedded SMILES string.

NoSequence

Returned if the molecule has large portion of its sequence unmapped to known monomers.

TooLarge

Returned if the molecule has more than 2048 heavy atoms and it is considered too large for HELM generation.

UnmappedFragments

Returns if there are any unmapped fragments in the molecule that would obstruct the generation of a valid HELM string without embedded SMILES.

See also

- *OEHelmGenerationOptions.GetAllowUnmatchedFragments* method

UnspecifiedAtomStereo

Returns if there is any unspecified stereo of a chiral atom in the molecule that would obstruct the generation of a valid HELM string without embedded SMILES.

See also

- *OEHelmGenerationOptions.SetAllowUnspecifiedStereo* method

UnspecifiedBondStereo

Returns if there is any unspecified stereo of a chiral bond in the molecule that would obstruct the generation of a valid HELM string without embedded SMILES.

See also

- *OEHelmGenerationOptions.SetAllowUnspecifiedStereo* method

OEHelmParsingReturnCode

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The *OEHelmParsingReturnCode* namespace contains codes representing the success or various issues of HELM string parsing when using the *OEHelmToMol* function.

See also

- *OEHelmParsingResult* class

Success

Represents that the HELM is **successfully** parsed.

HelmInvalidComponent

HelmInvalidVersion

HelmSyntaxError

InvalidConnectionChainId

The HELM connection component references a chain that does not exist.

InvalidConnectionMonomerIdx

The HELM connection component references a monomer that does not exist.

InvalidConnectionRGroupIdx

The HELM connection component references a chain that does not exist.

InvalidEmbeddedSmiles

The HELM contains embedded SMILES string that can not be parsed and converted into a monomer.

InvalidMonomerConnection

The HELM contains monomer connections that are invalid.

NotSupported

The HELM contains features that is not supported yet in the current implementation.

UnrecognizedMonomerCode

The HELM string includes a monomer code that is not present in the selected code sets.

OERxnAutomapMethod

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The *OERxnAutomapMethod* namespace encodes symbolic constants representing the mapping method to use for the reaction automapping activity. This is just a placeholder for future mapping methods.

> **See also**
>
> - *OERxnAutomapper* class

This namespace contains the following constants:

DEFAULT

This value requests the default mapping based algorithm, currently MCSMapping.

MCSMapping

This value requests an MCS mapping based algorithm.

OERxnAutomapStatus

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The *OERxnAutomapStatus* namespace encodes symbolic constants representing the status returned from reaction automapping.

> **See also**
>
> - *OERxnAutomapper* class

This namespace contains the following constants:

Failure

This status indicates a general failure attempting to automap a reaction.

InvalidComponents

This status indicates that the input reaction is missing either reactants or products and thus cannot have mapping correspondence assigned.

InvalidMethod

This status indicates that an invalid mapping method was requested.

NonReactionInput

This status indicates that the input structure is not identified as a reaction.

See also

- *IsRxn*

Success

This status indicates success.

OEMonomerType

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The *OEMonomerType* namespace contains the following constants:

Default

The default polymer type is *OEMonomerType.Backbone*.

Backbone

String representation: `Backbone`

Monomer with multiple connection point. In case of a peptides, these monomers are used to build the backbone.

Terminal

Monomer with one attachment point. In case of a peptides, these monomers are used at the C and N-terminal.

String representation: `Terminal`

Undefined

See also

- *OEGetMonomerType* function

- *OEHasMonomerType* functor
- *OEMonomerData* class
- *OEStringToMonomerType* and *OEMonomerTypeToString* functions

OEMonomerValidationReturnCode

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The *OEMonomerValidationReturnCode* namespace contains codes representing the success or various issues of monomer validation when using the *OEIsValidMonomerData* function.

See also

- *OEMonomerData* class
- *OEMonomerValidationResult* class

It contains the following constants:

Success

Represents that the monomer is **valid**.

InvalidMonomerCode

The monomer code is invalid:

- empty string or more the 20 character long
- contains character other than: alphanumerical, - (dash), _ (underscore), (or) or , (comma)
- UNK is a reserved name and should nt be used as a monomer code

InvalidMonomerRGroups

If the monomer has any invalid connection point defined my R-group map index.

InvalidMonomerSmiles

The monomer smiles is invalid, if the *OEParseSmiles* function return false.

InvalidMonomerType

The monomer type is invalid or does not match to the type returned by the *OEGetMonomerType* function.

InvalidPolymerType

The polymer type is invalid or does not match to the type returned by the *OEGetPolymerType* function.

MissingMonomerCode

If the monomer has not code specified for any code-set.

MissingMonomerRGroups

If the monomer has no connection point defined my R-group map index.

MissingMonomerSmiles

If no smiles specified for the monomer structure.

UnspecifiedAtomStereo

A monomer unspecified stereo at a chiral atom is considered invalid.

UnspecifiedBondStereo

A monomer unspecified bond at a chiral bond is considered invalid.

OEPolymerType

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

The *OEPolymerType* namespace contains the following constants:

Default

The default polymer type is *OEPolymerType.Peptide*.

Chem

Represents monomers with generic polymer type.

String representation: `Chem`

Peptide

Represents peptidic monomers that can be used to define large peptide molecules.

String representation: `Peptide`

Undefined

See also

- *OEGetPolymerType* function
- *OEHasPolymerType* functor
- *OEMonomerData* class
- *OEStringToPolymerType* and *OEPolymerTypeToString* functions

OESMARTSAtomFlag

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

This namespace contains constants representing various supported SMARTS pattern constraints for atoms.

AAmap

This constant requests the explicit inclusion of non-zero mapping number information.

AtomType

AnyAtomType

IgnoreArom

These constants control the output of atom type information. The returned atom SMARTS string will only be `[]` delimited if one of `AnyAtomType` or `AtomType` are specified which allows using non-atype flags to build up SMARTS qualifiers for alternate custom SMARTS generation activities.

`AtomType` will encode the atom symbol of the atom type which is downcased for aromatic atoms. `AnyAtomType` will encode `a,A` for any atom type, depending on aromatic atom state. `IgnoreArom` will encode just the atomic number value of the atom.

Charge

Charge0

These constants control the output of atom formal charge information. For `Charge`, only non-zero charge information is output, and for `Charge0`, information is output explicitly regardless of value.

Chirality

This constant requests the encoding of atom parity information via the `@`, `@@` designation on the atom. Note that this information requires the presence of the neighbor atoms in the SMARTS query to be a meaningful mapping constraint. Additionally, the encoded parity state is explicitly dependent on the ordering of neighbor atoms, requiring suppression of any reordering of atoms when generating the full molecule SMARTS.

See also

- *OESMILESFlag*
- *OECreateSmartsString*

Connectivity

This constant controls the output of the connectivity for the atom, see *GetDegree*.

DegreeCount

This constant controls the output of the explicit degree for the atom, see *GetExplicitDegree*.

HCount

HCount0

These constants control the output of atom total hydrogen count information. For `HCount`, only non-zero information is output, and for `HCount0`, information is output explicitly regardless of value. See *GetTotalHCount*.

Hybridization

This constant controls the output of nonzero hybridization for the atom, see *GetHyb*. The hybridization of atoms should be set up via *OEAssignHybridization* for this flag to be active.

ImplicitHCount

ImplicitHCount0

These constants control the output of implicit hydrogen counts for the atom which can be assigned via *OEAssignImplicitHydrogens*, but may already be assigned depending on how the molecule was instantiated and from what source. For `ImplicitHCount`, only non-zero information is output, and for `ImplicitHCount0`, information is output explicitly regardless of value. See *GetImplicitHCount*.

Isotope

This constant controls whether isoptopic information on the atom should be encoded.

LeavingGroup

This constant is experimental.

RingBondCount

This constant controls the output of the ring bond count for the atom.

Valence

This constant controls the output of the valence value for the atom, see *GetValence*.

AtomAll

This constant is a collection of all available flags. Equivalent to

```
(AtomType | IgnoreArom | AnyAtomType | DegreeCount | HCount | HCount0 | Charge | Charge0
| RingBondCount | Valence | Connectivity | AAmap | LeavingGroup | Isotope | Chirality |
Hybridization | ImplicitHCount | ImplicitHCount0)
```

DEFAULT

AtomDefault

These two constant values equivalently request that no atom pattern constraints are generated.

AtomExact

This constant provides a collection of atom pattern constraints suitable for exact-match searching. Equivalent to `(AtomType | Isotope | RingBondCount | Valence | HCount0 | Connectivity | Charge0)` These flags are not included, as they require consideration of neighbor atoms, or need setup perception activity to be meaningful, but should be carefully considered for augmenting an exact-match context: `Chirality`, `Hybridization`, `ImplicitHCount`, `ImplicitHCount0`.

OESMARTSBondFlag

OELigandEnergy

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

This namespace contains constants representing various supported SMARTS pattern constraints for bonds.

BondType

AromBondType

AnyBondType

For non-aromatic bond types, `BondType` exports the explicit bond order/type, `-,=,#`. For aromatic bonds types and `AromBondType` specified, an aromatic bond type is exported, `:`. If `AnyBondType` is specified, a generic any bond type is exported, ~.

BondTopo

For non-aromatic bonds, ring/chain constraints are exported.

Chirality

This constant requests the encoding of bond cis/tran parity information via forward and backward `/` designations on the bond neighbors. Note that this information requires the presence of the neighbor atoms of the geometric double bond in the SMARTS query to be a meaningful mapping constraint. Additionally, the encoded cis/trans parity state is explicitly dependent on the ordering of neighbor atoms, requiring suppression of any atom reordering when generating the full molecule SMARTS, as well as requesting *`OESMILESFlag.BondStereo`* during the molecule SMARTS generation.

See also

- *OESMILESFlag*
- *OESMILESFlag.BondStereo*
- *OECreateSmartsString*

BondAll

This constant is a collection of all available flags. Equivalent to `(BondType | AromBondType | AnyBondType | BondTopo | Chirality)`

DEFAULT

BondDefault

These two constant values equivalently request that no bond pattern constraints are generated.

BondExact

This constant provides a collection of bond pattern constraints suitable for exact-match searching. Equivalent to `(BondType | AromBondType | BondTopo)` This flag is not included, as it requires consideration of neighbor atoms to the cis/trans bond but should be carefully considered for augmenting an exact-match context: `Chirality`.

OEAroModel

This namespace contains constants representing various supported aromaticity when calling the *OEAssignAromaticFlags* function.

See also

- *Aromaticity Models in OEChem TK* section

Daylight

This constant represents the Daylight aromaticity model.

OpenEye

This constant represents the default *OpenEye* aromaticity model.

Tripos

This constant represents the Tripos aromaticity model.

MMFF

This constant represents the MMFF aromaticity model.

MDL

This constant represents the MDL aromaticity model.

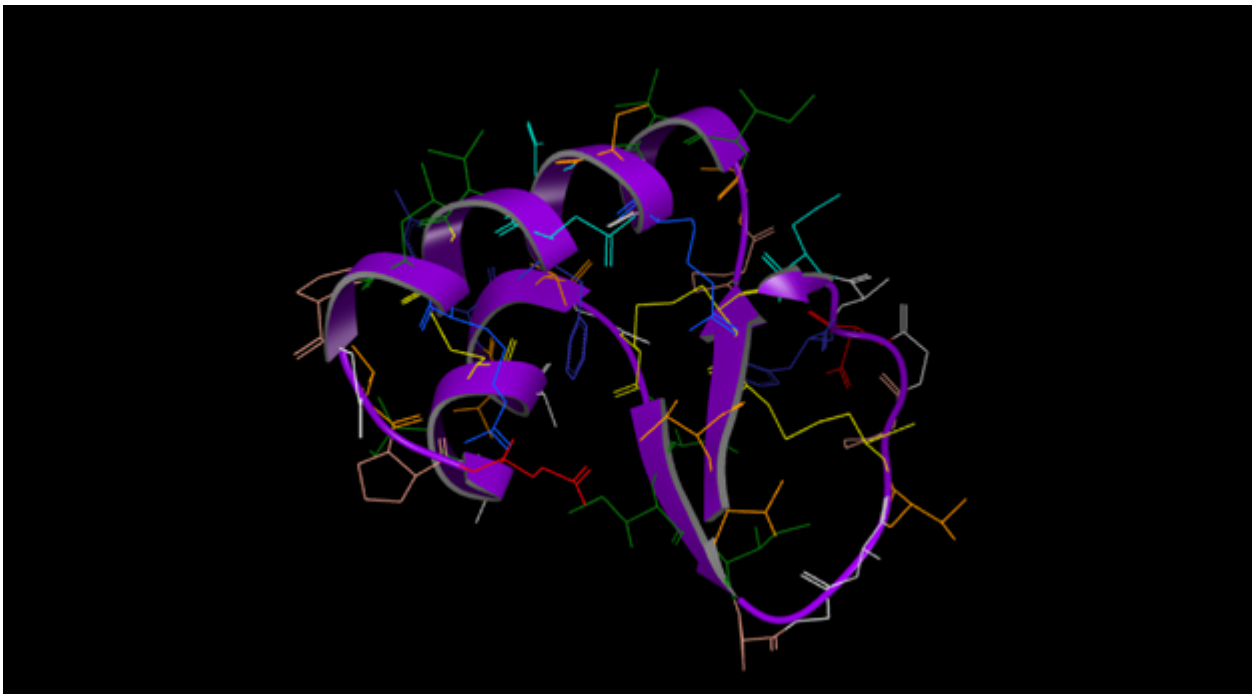
OEAtomColorScheme

The OEAtomColorScheme namespace encodes symbolic constants utilized to specify the color scheme of the atoms in an *OE3DMolStyle*. The constants can be used as an argument when constructing a *OEMolStyleColorer*, which can be used as an argument for the `OE3DMolStyle.SetAtomColorer` function in *OE3DMolStyle*.

This namespace contains constants.

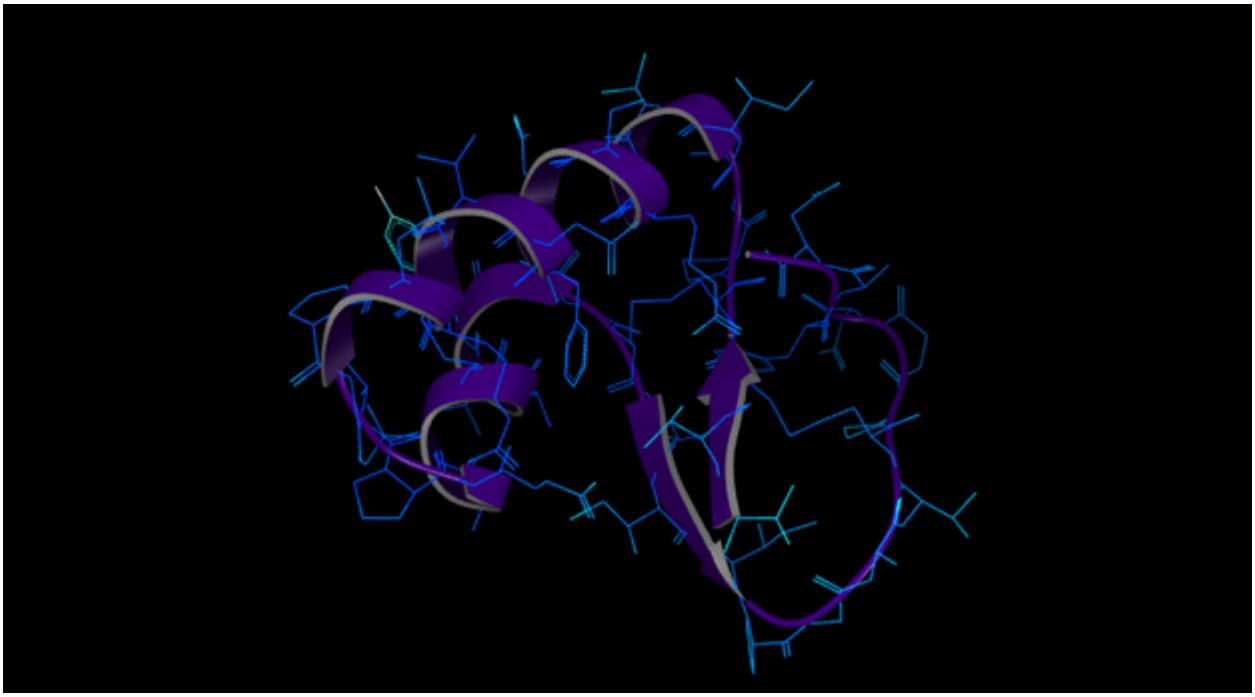
AltLocation

Amino



BEGIN

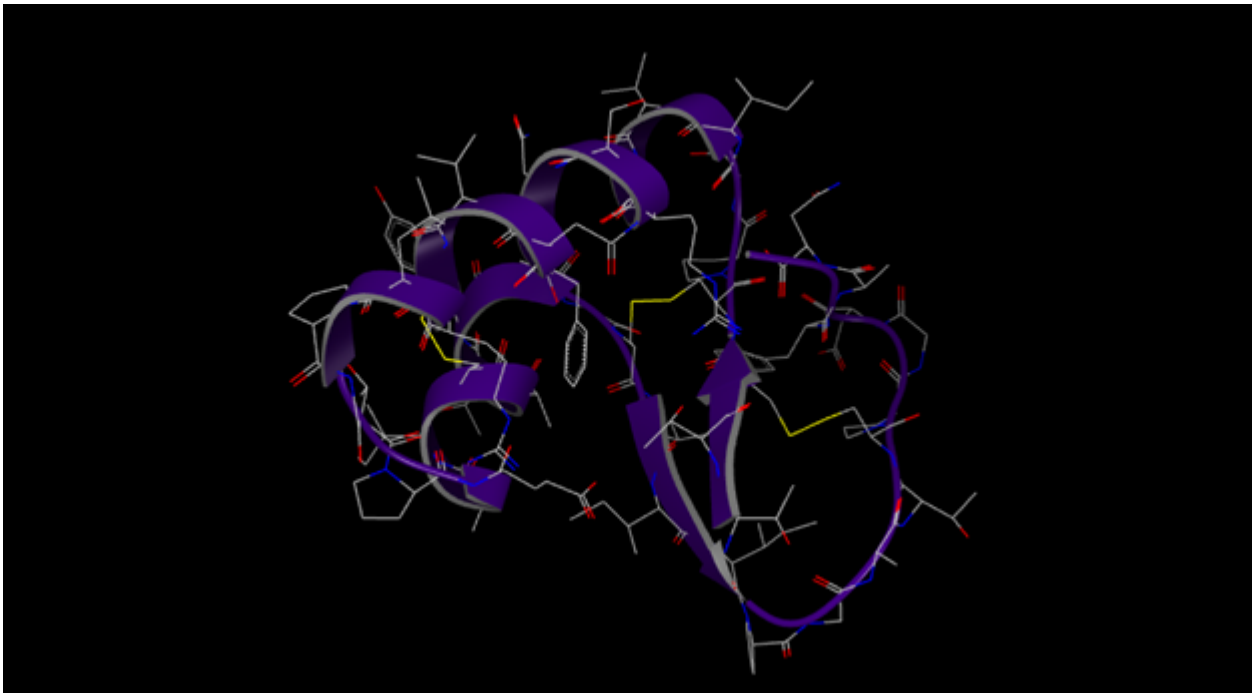
BFactor



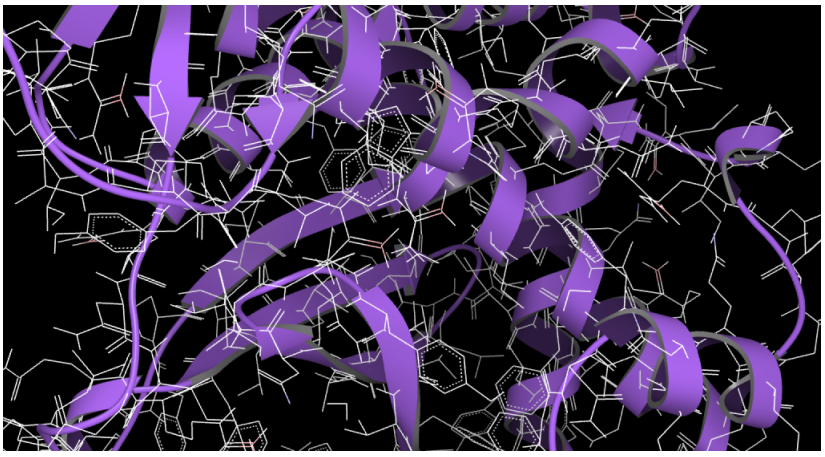
Chain

END

Element



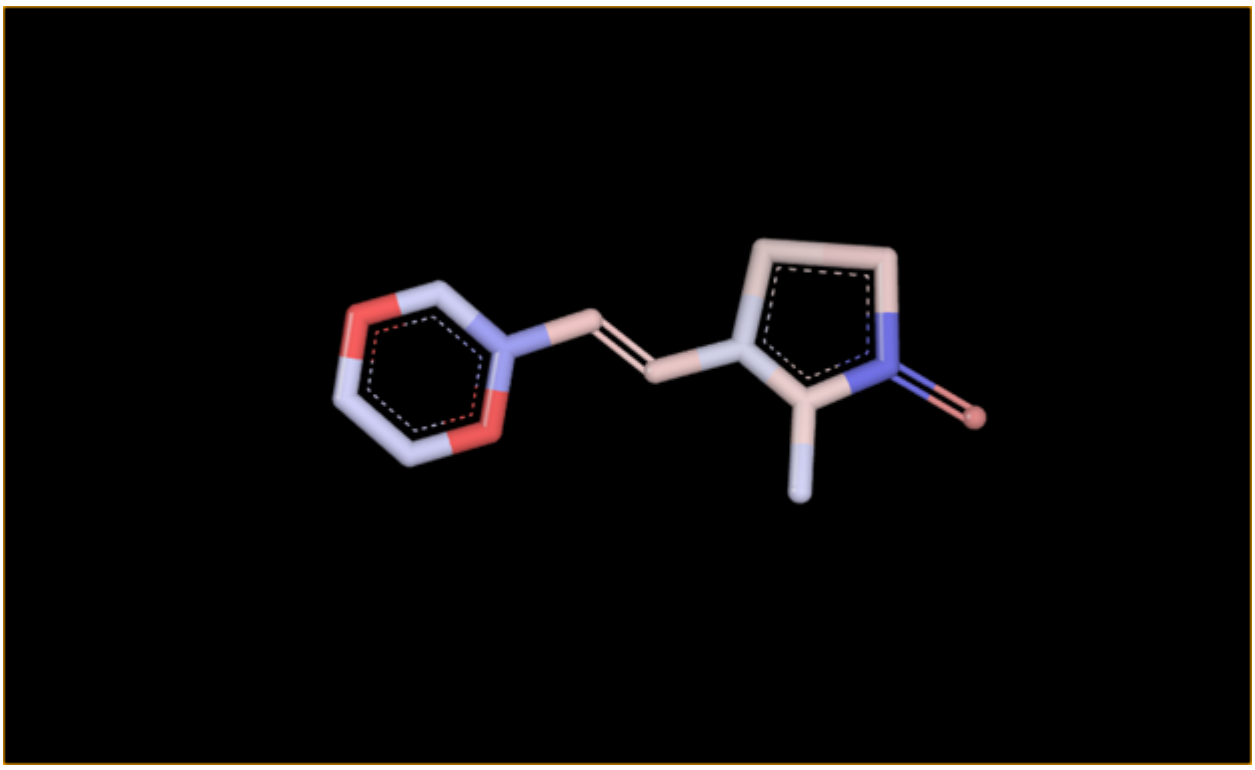
FormalCharge



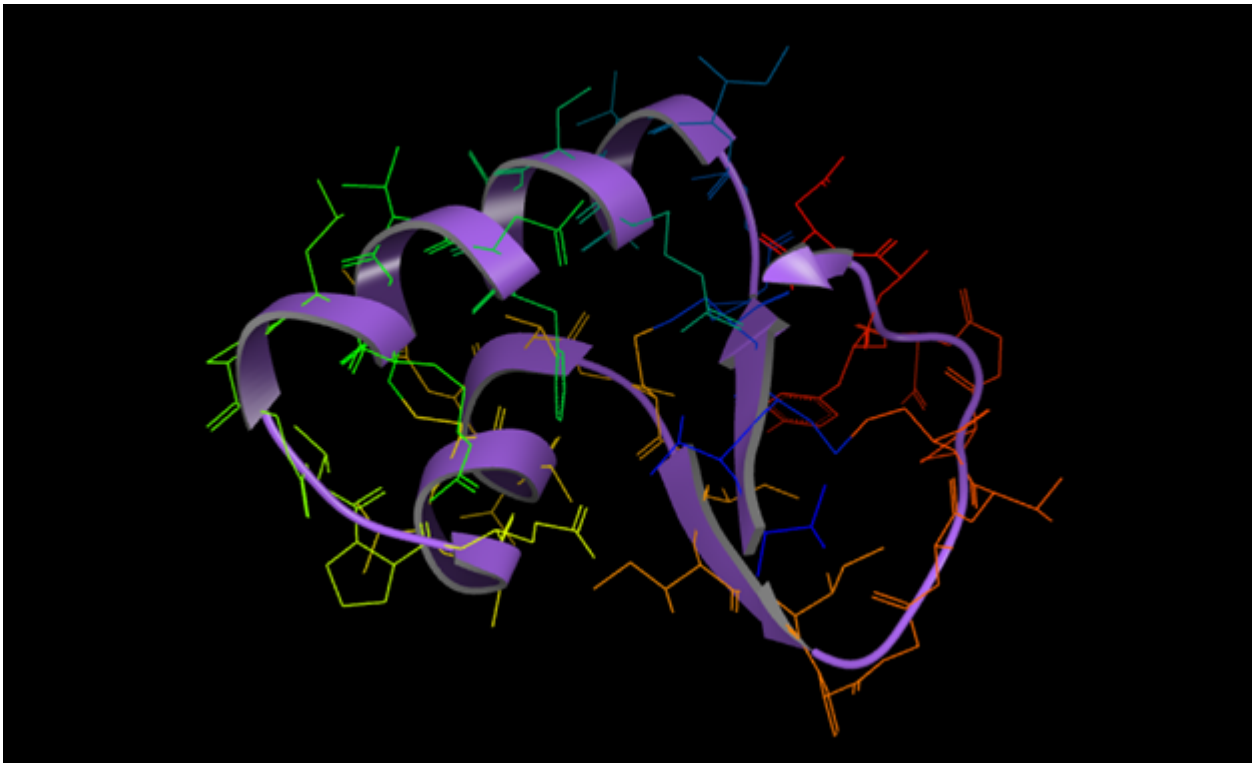
GenericData

Occupancy

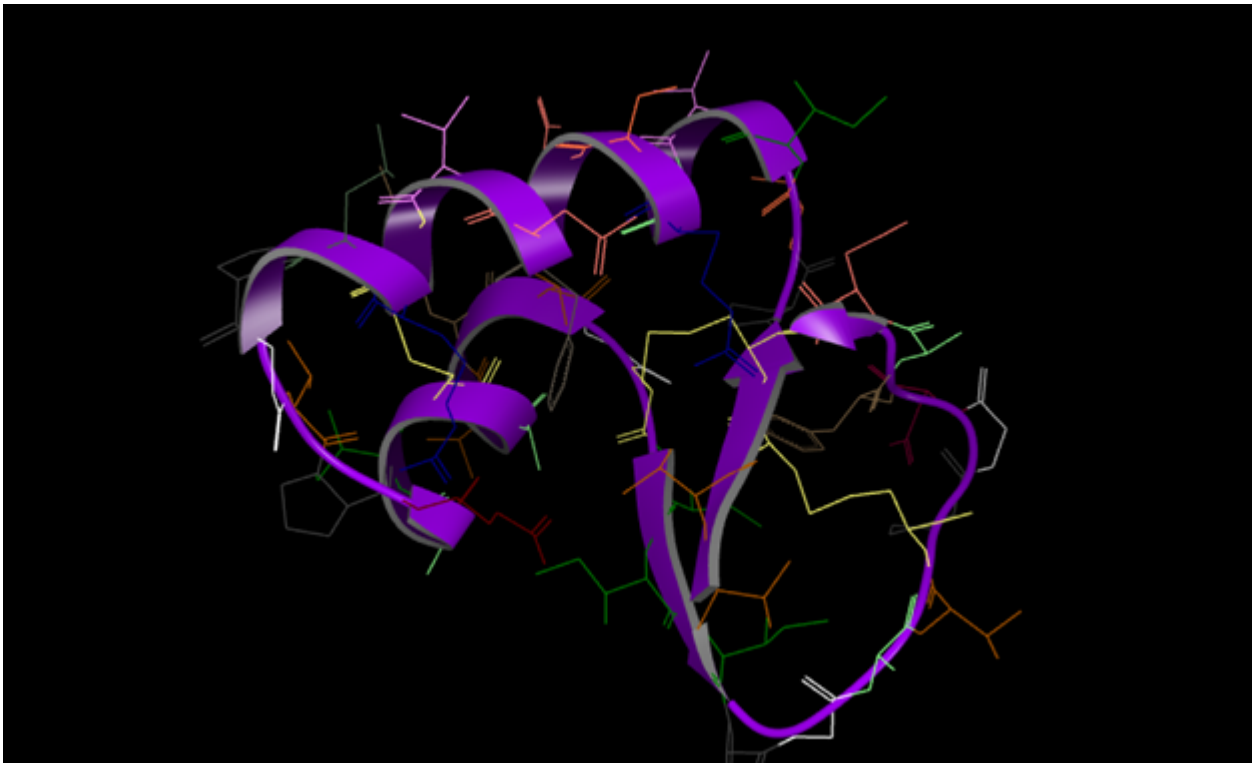
PartialCharge



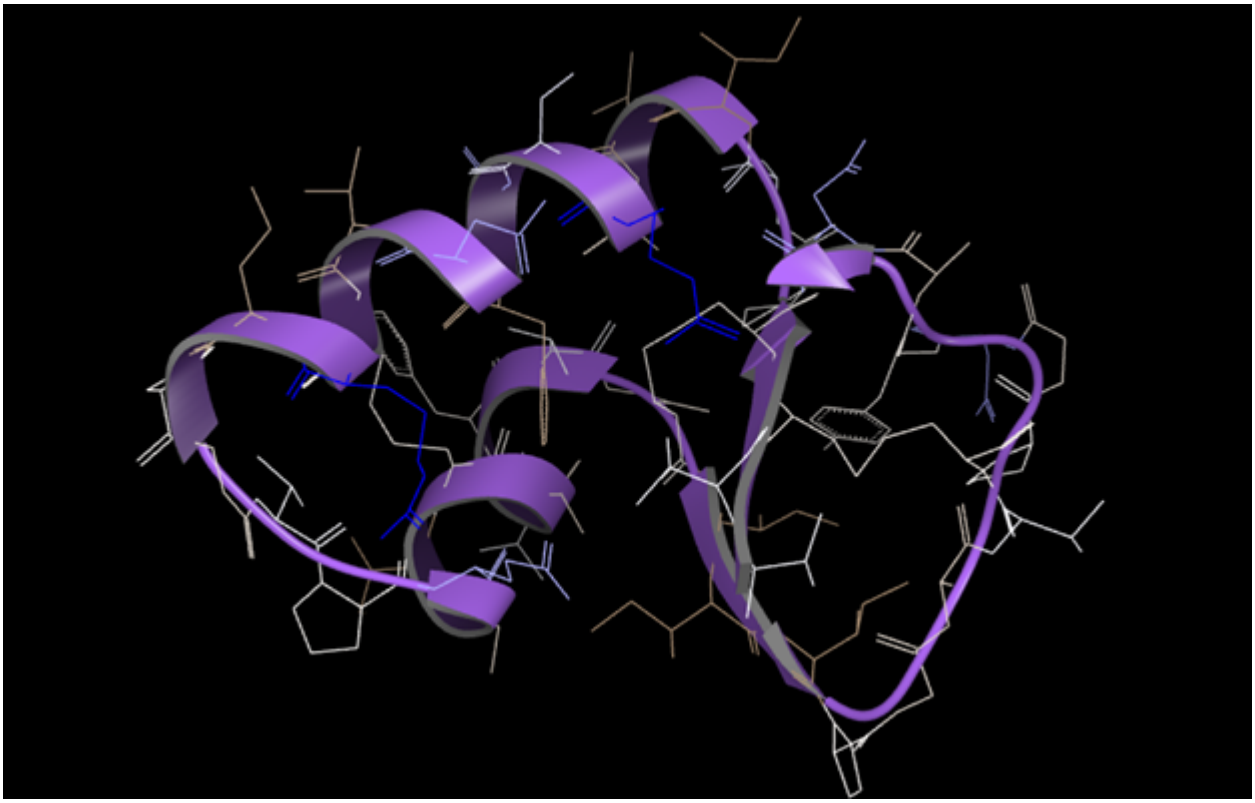
ProteinSequence



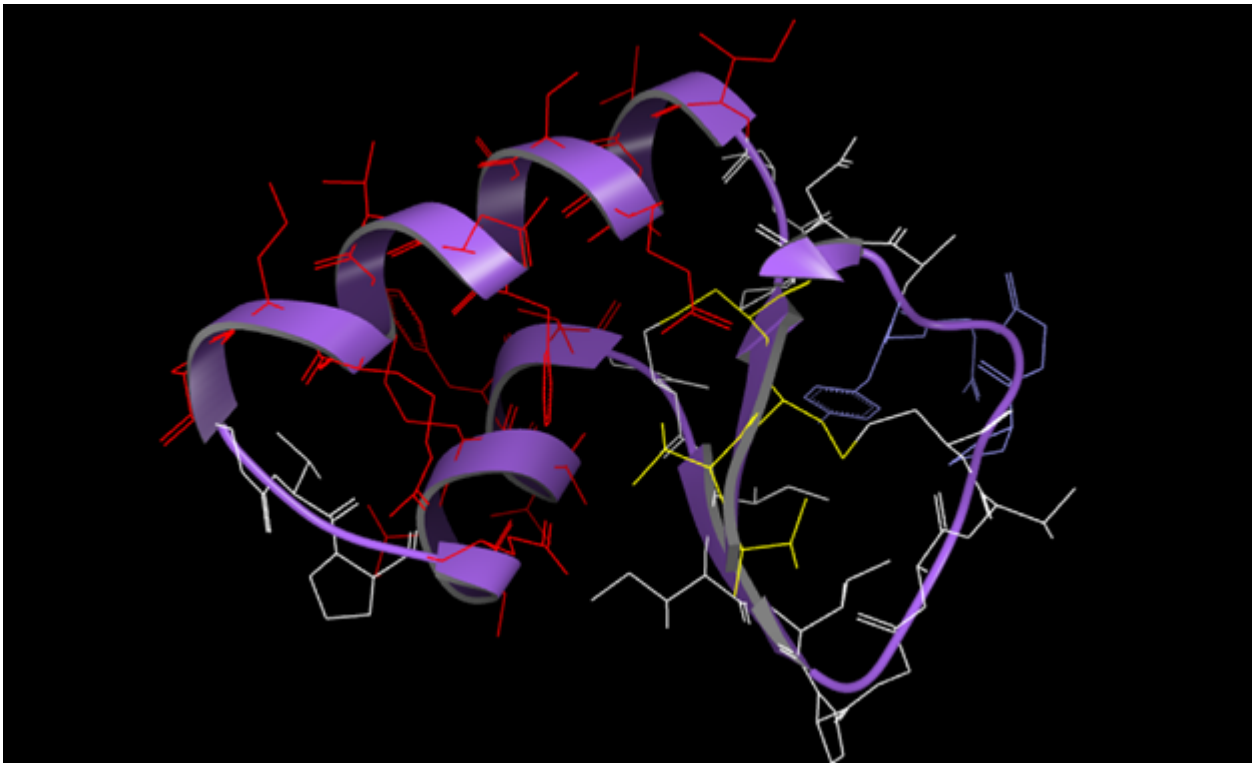
Residue



ResidueHydrophobicity

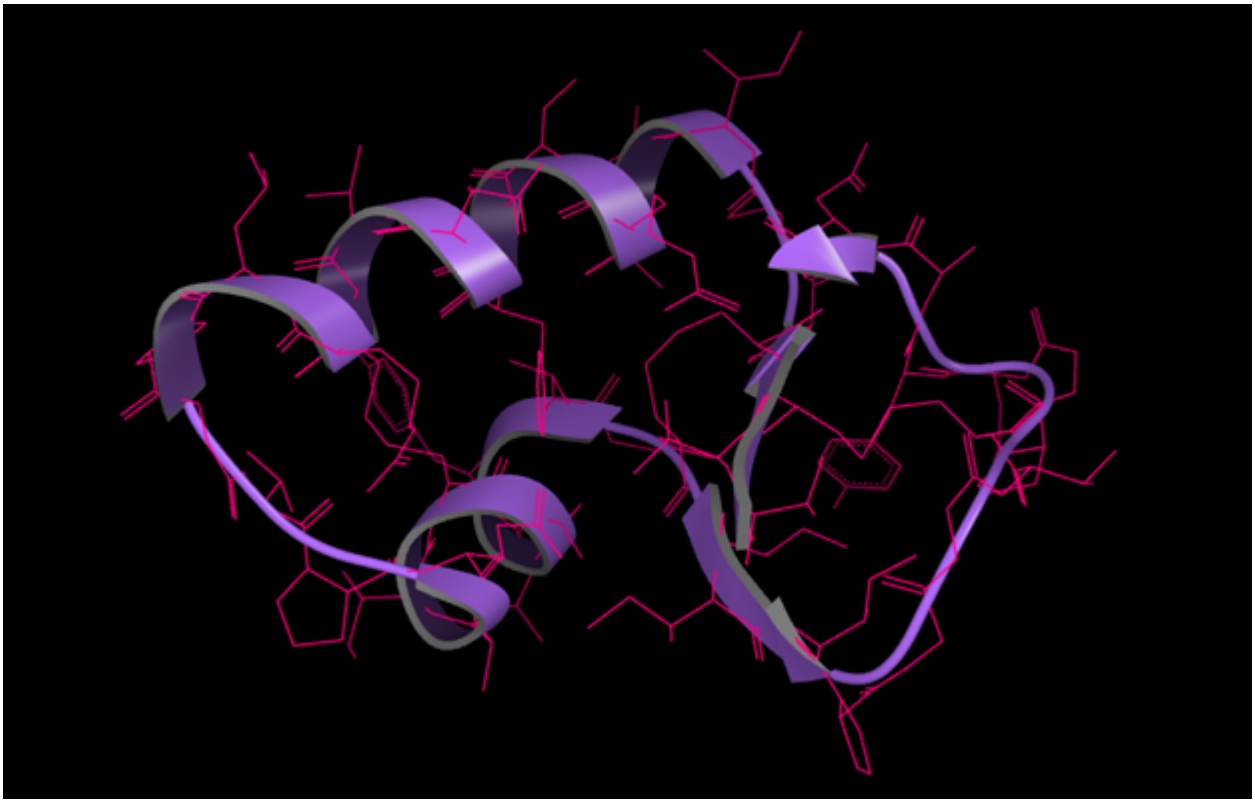


SecondaryStructure



Shapely

Solid



OEAtomLabelScheme

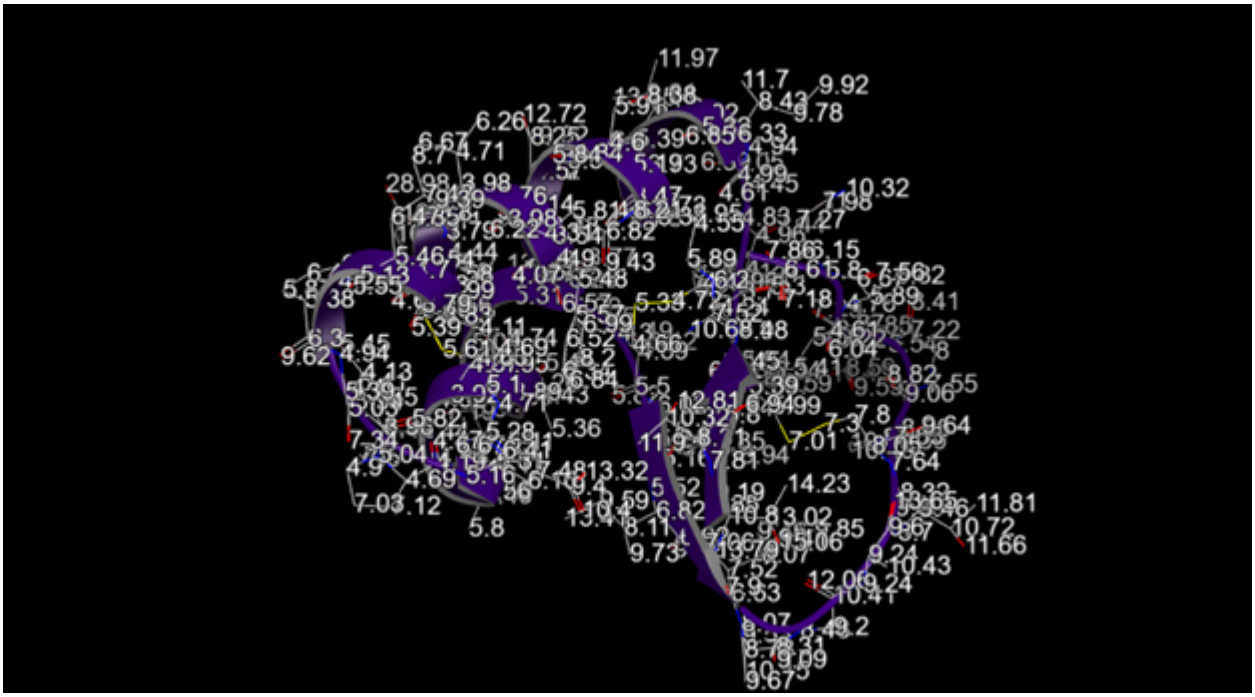
The OEAtomLabelScheme namespace encodes symbolic constants utilized to specify the labeling scheme of the atoms in a *OE3DMolStyle*. The constants can be used as an argument for the `OE3DMolStyle.SetAtomLabelType` function in *OE3DMolStyle*.

This namespace contains constants.

AltLocation

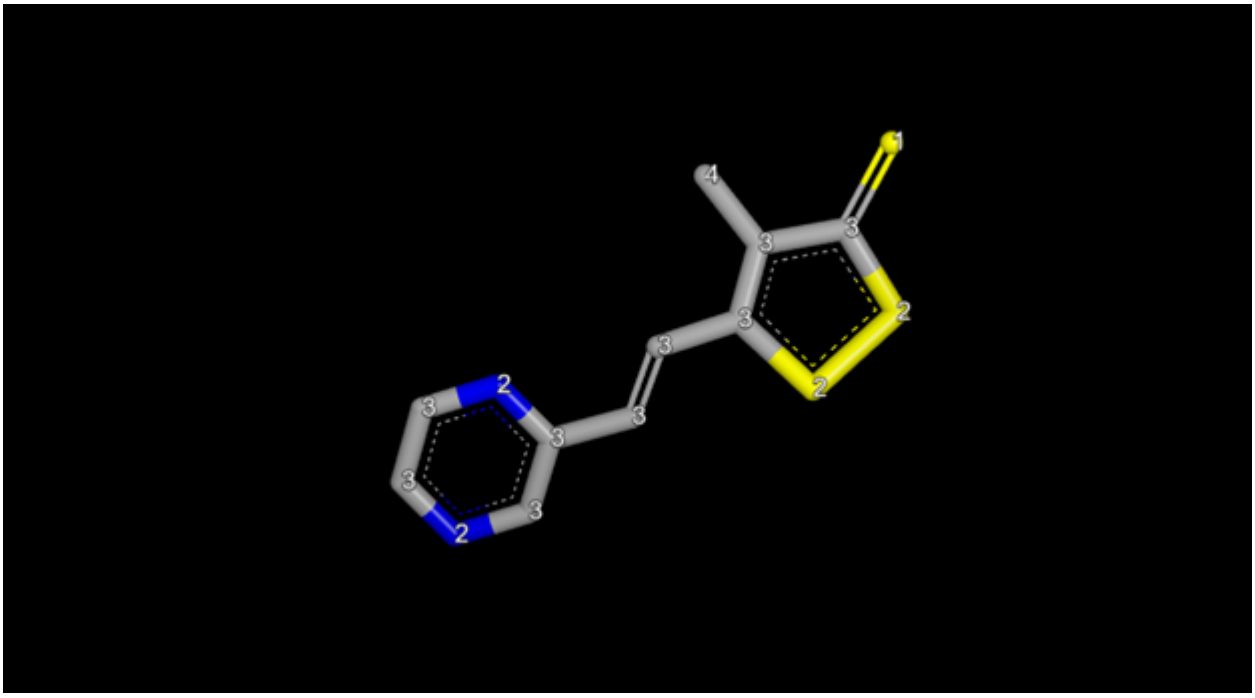
BEGIN

BFactor



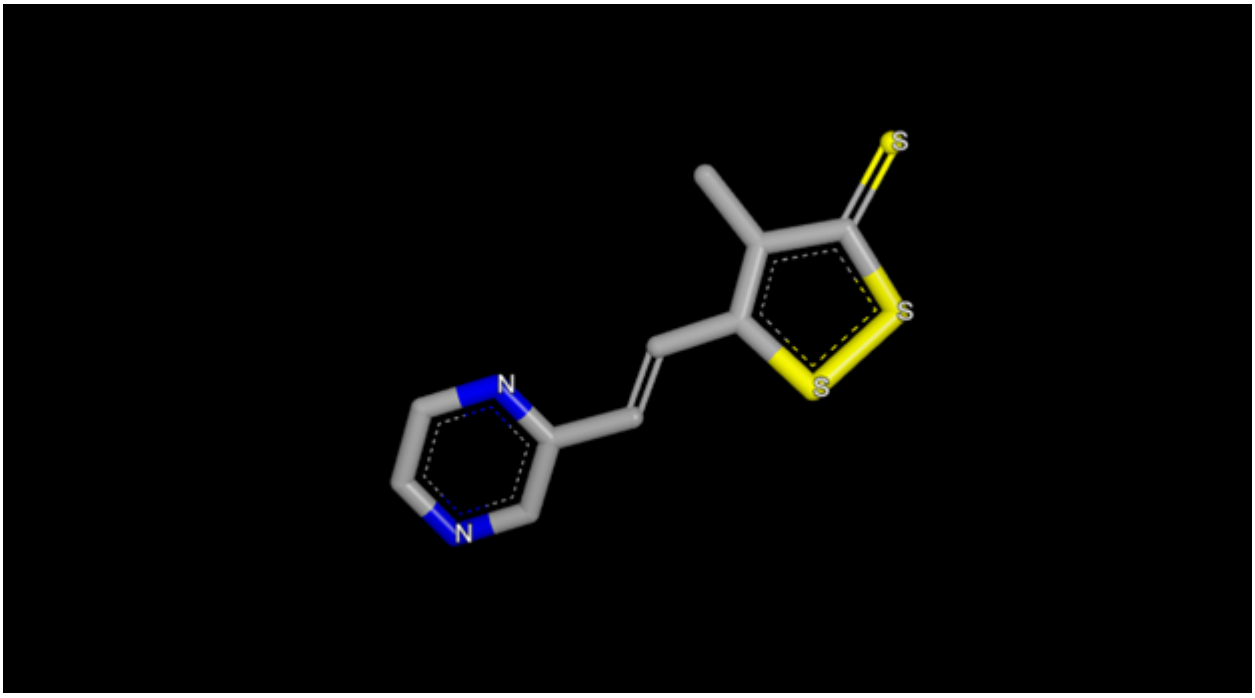
Chirality

Degree

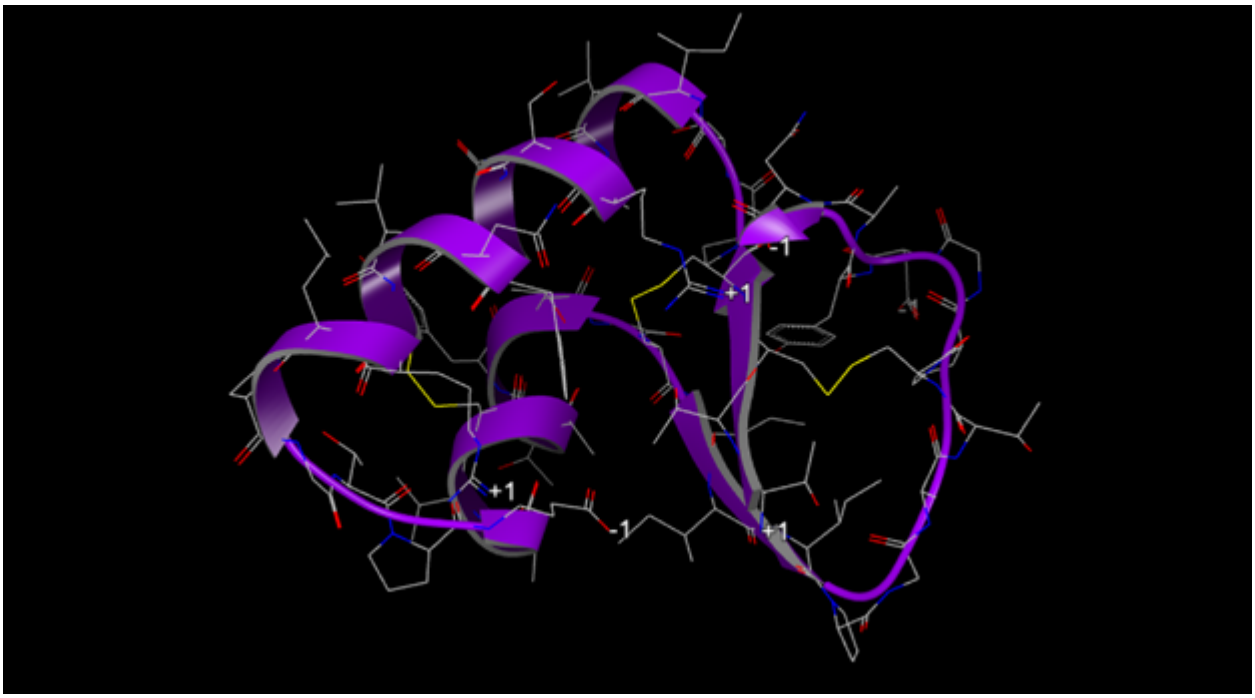


END

Element



FormalCharge

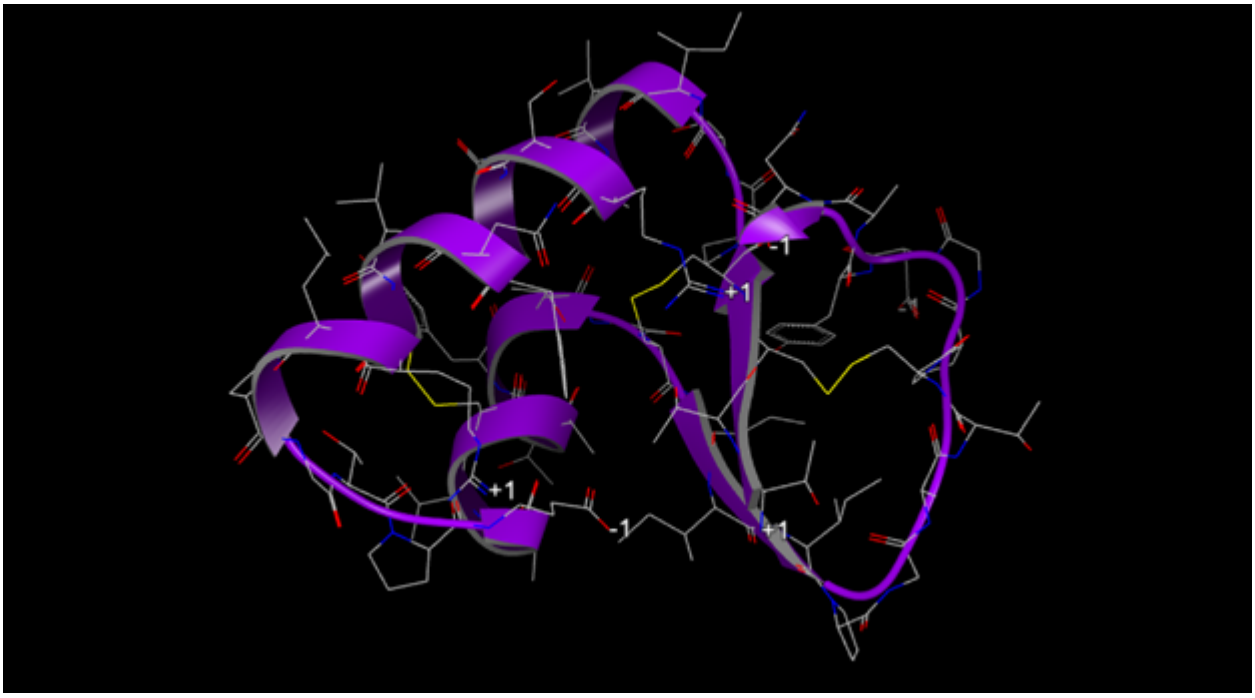


GenericData

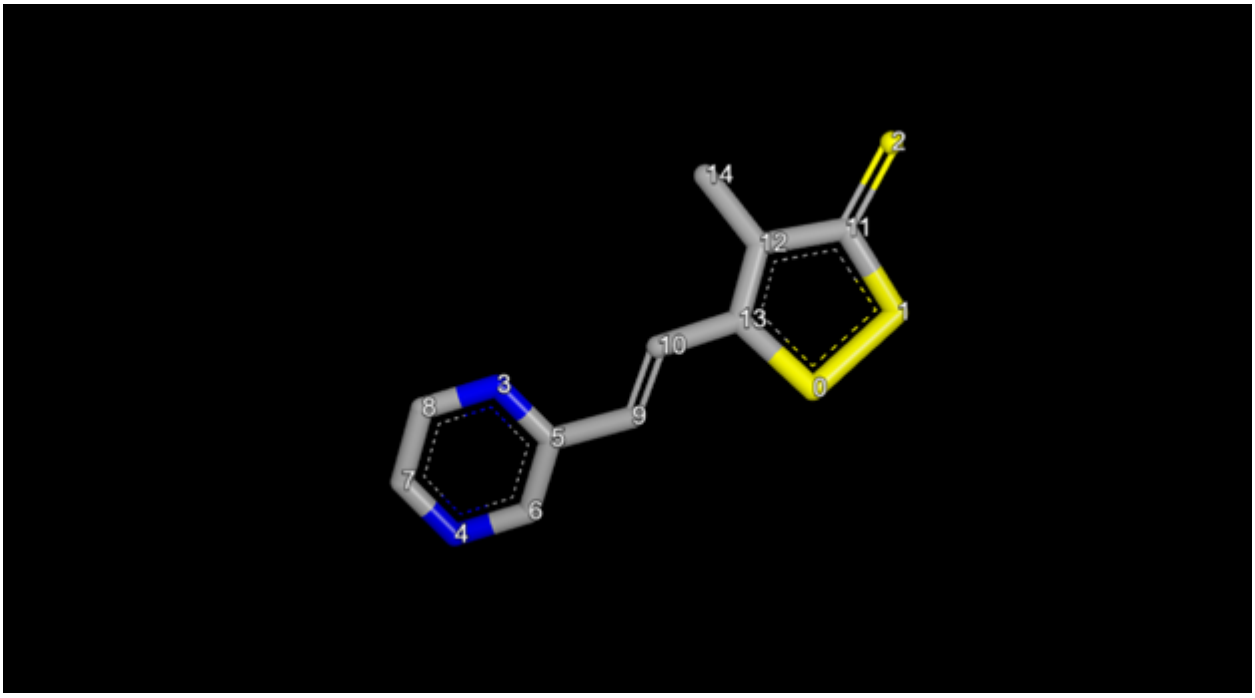
HasStereoSpecified

Hybridization

ImplicitHCount

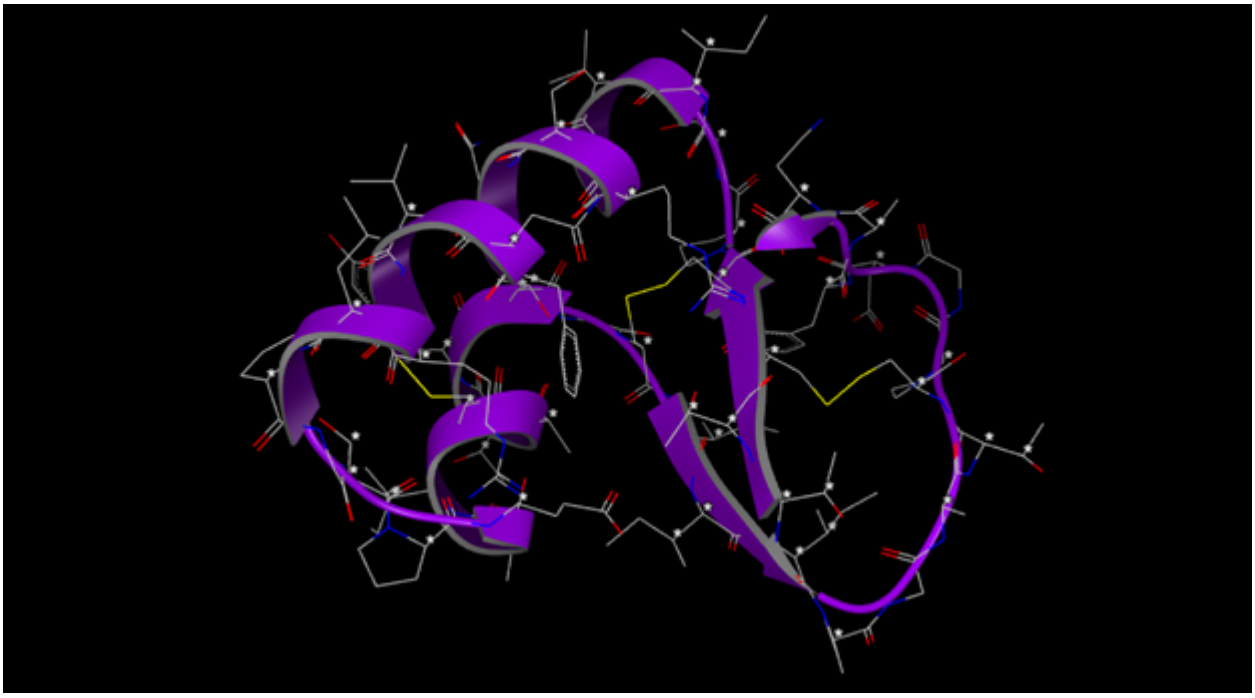


Index

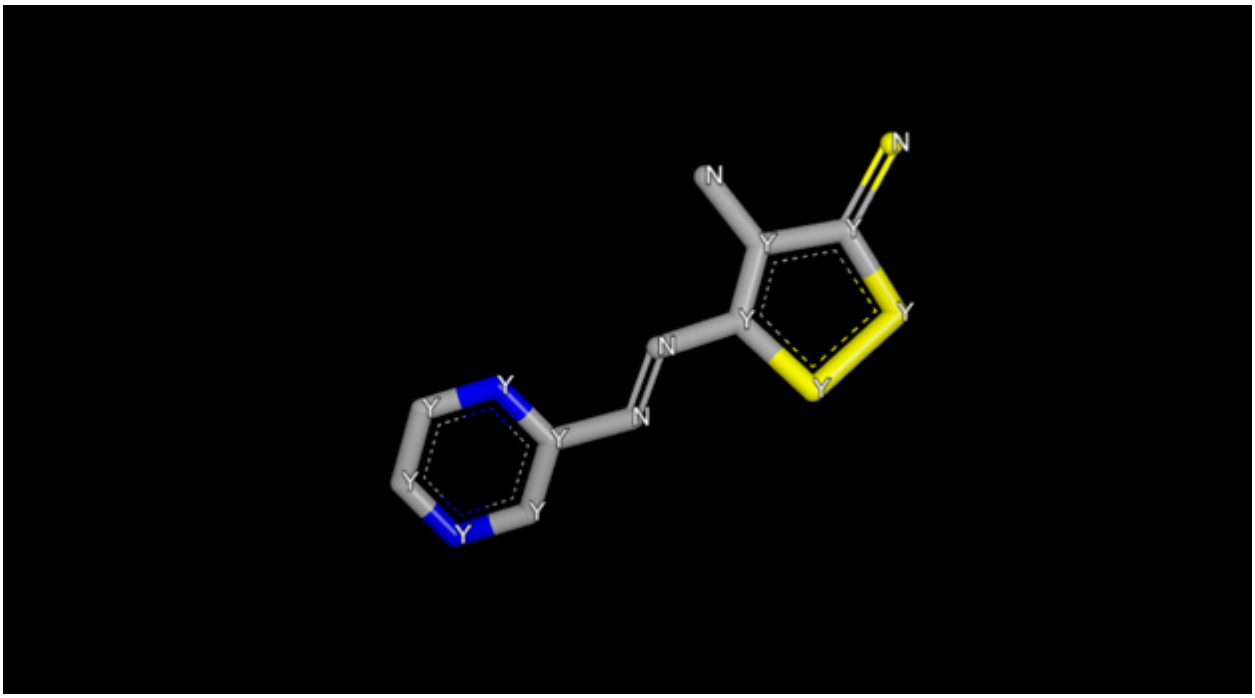


IntegerType

IsChiral



IsInRing



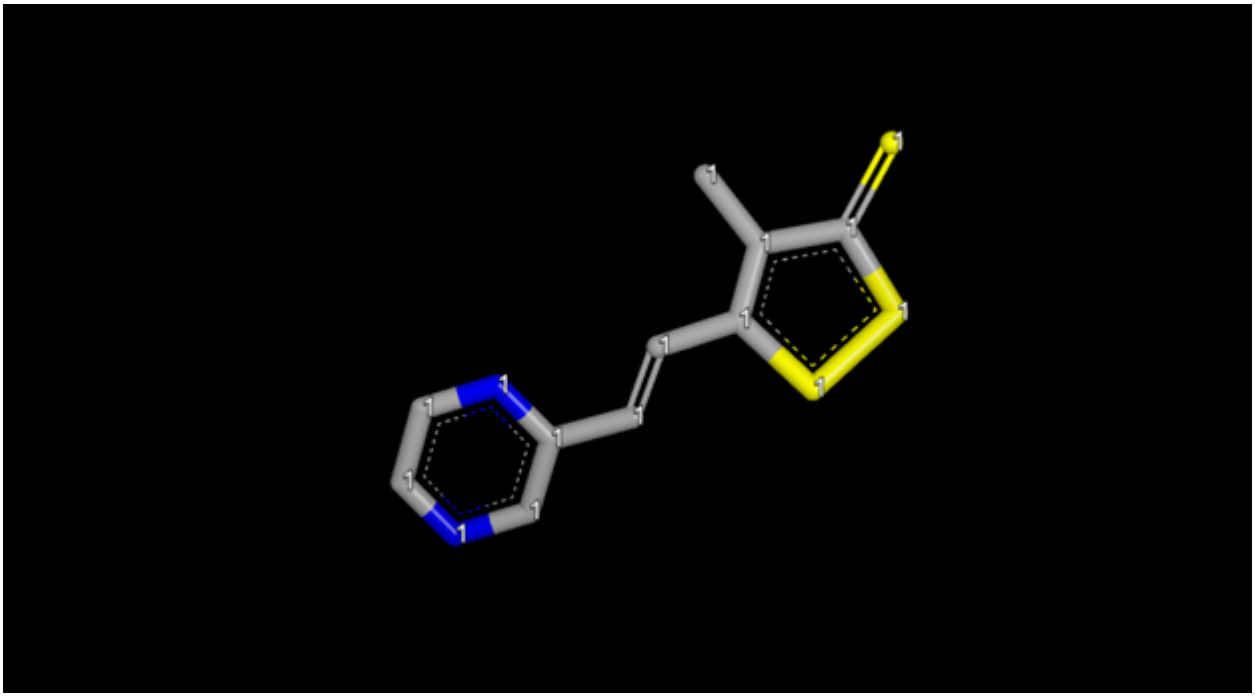
Isotope

MMFFType

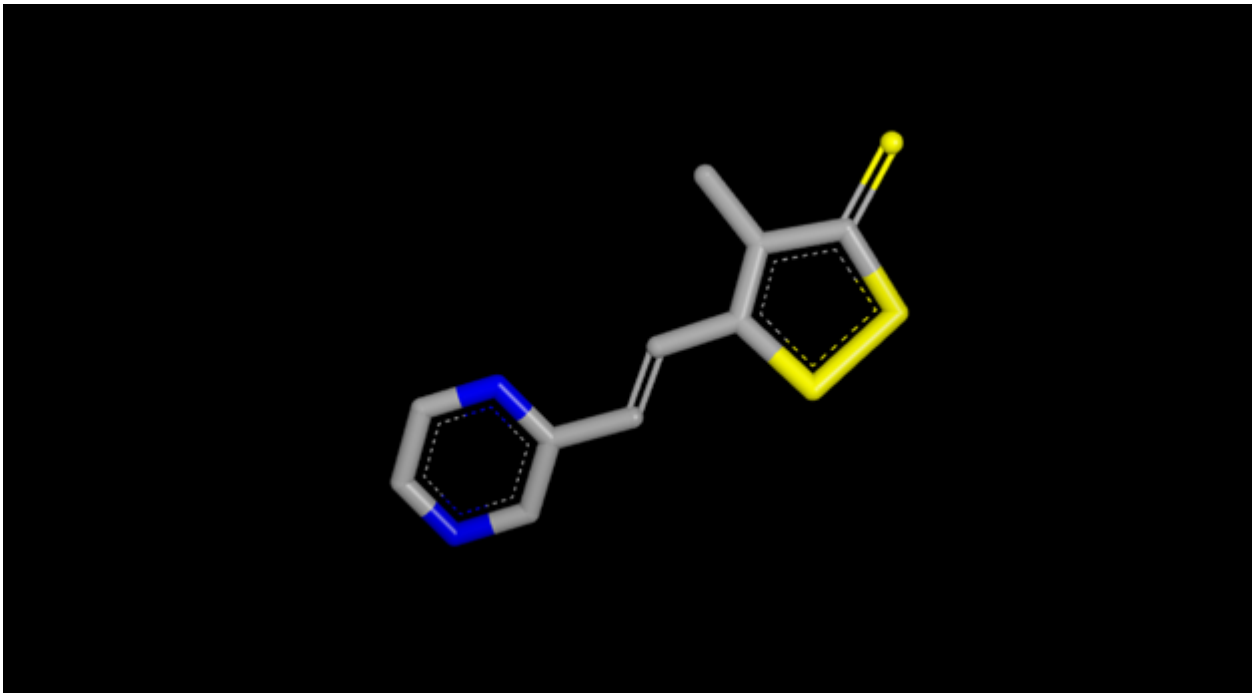
MapIdx

OEChemIdx

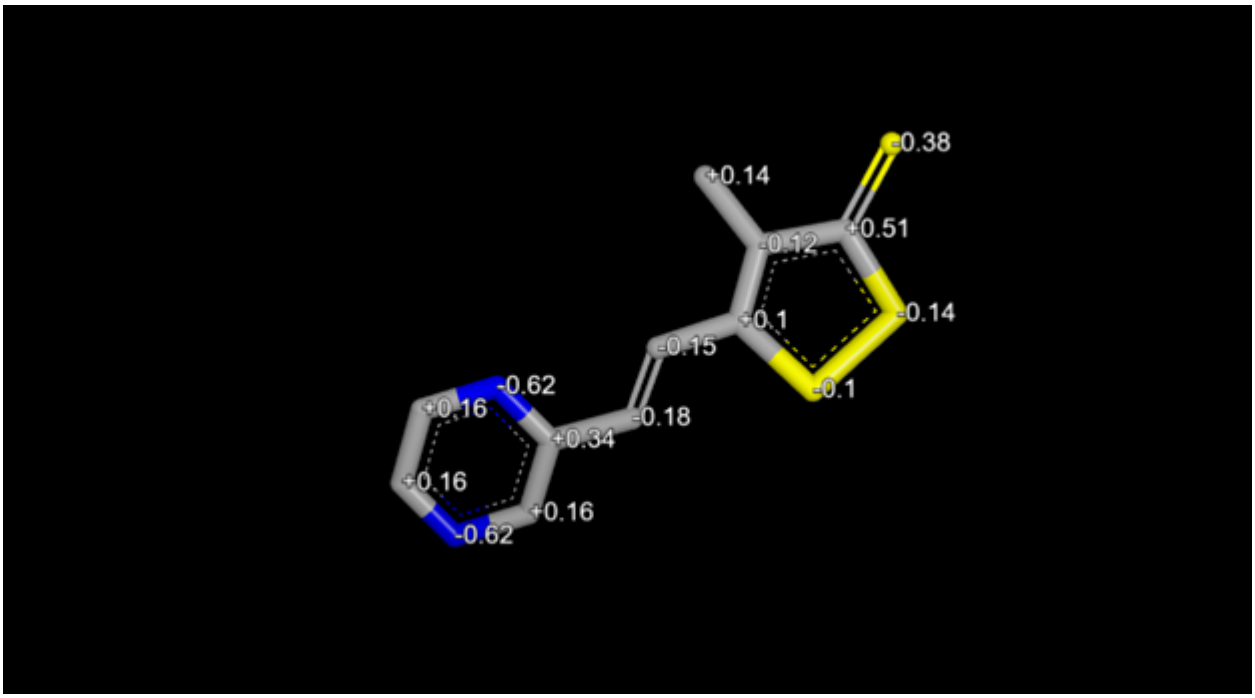
Occupancy



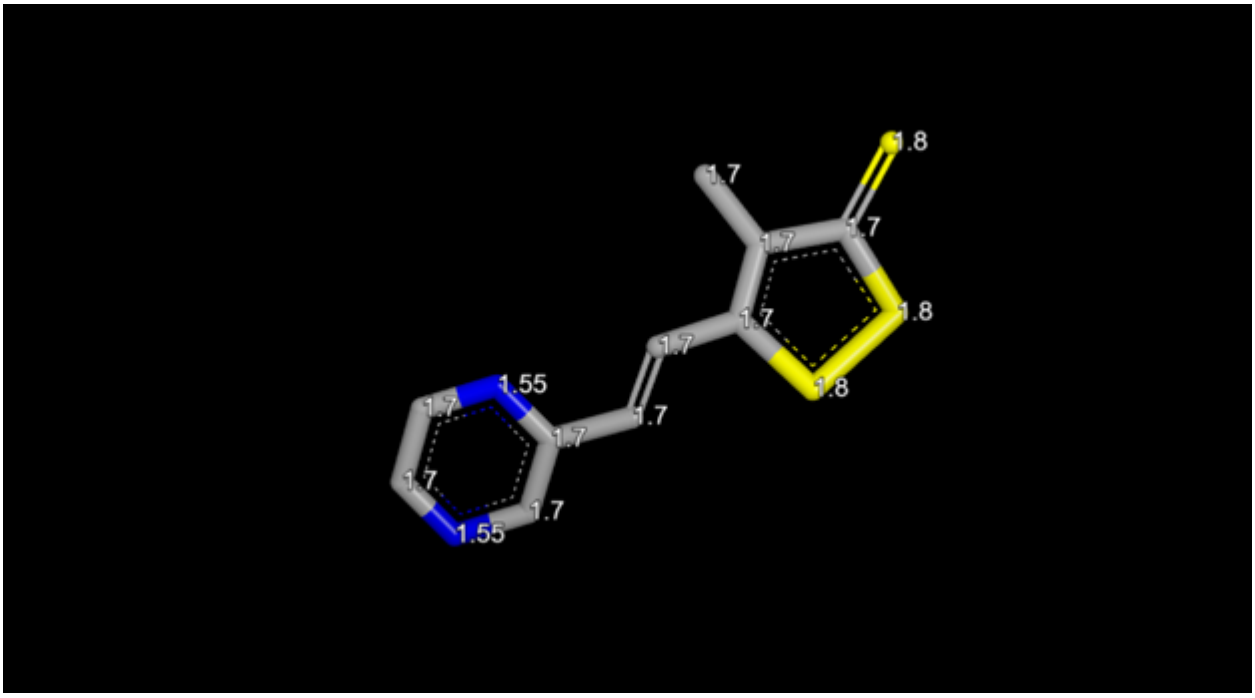
Off



PartialCharge



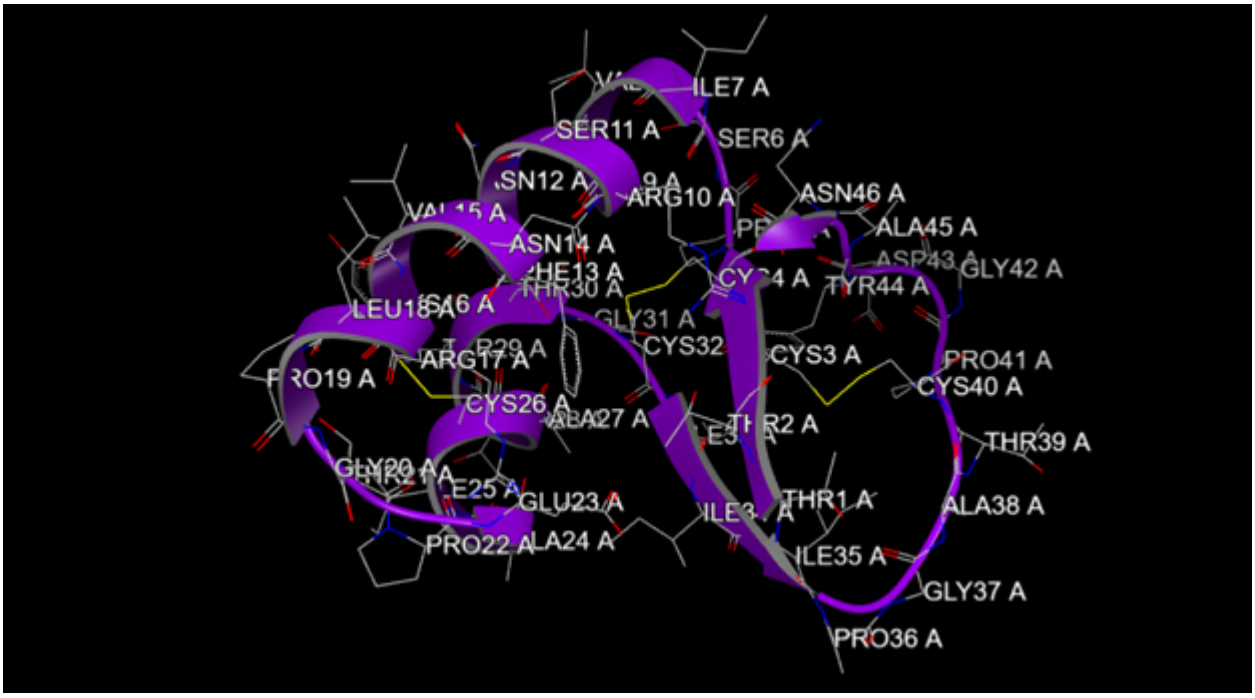
Radius



ResidueInfo



ResidueInfoCA



Symmetry

Type

User

OEAtomStereo

See also

- stereo atom
- *OEAtomBase.GetStereo* method
- *OEAtomBase.SetStereo* method
- *OEAtomBase.HasStereoSpecified* method

This namespace contains the following constants:

All

Represents all atom stereochemistry classes.

Left

Atom stereochemistry value of the *OEAtomStereo.Tetra* class.

LeftHanded

Same as the *OEAtomStereo.Left* constant.

Right

Atom stereochemistry value of the *OEAtomStereo.Tetra* class.

RightHanded

Same as the *OEAtomStereo.Right* constant.

Tetra

Atom stereochemistry class.

See also

OEAtomBase.SetStereo function

Tetrahedral

Same as the *OEAtomStereo.Tetra* constant.

Undefined

Represents undefined atom stereochemistry.

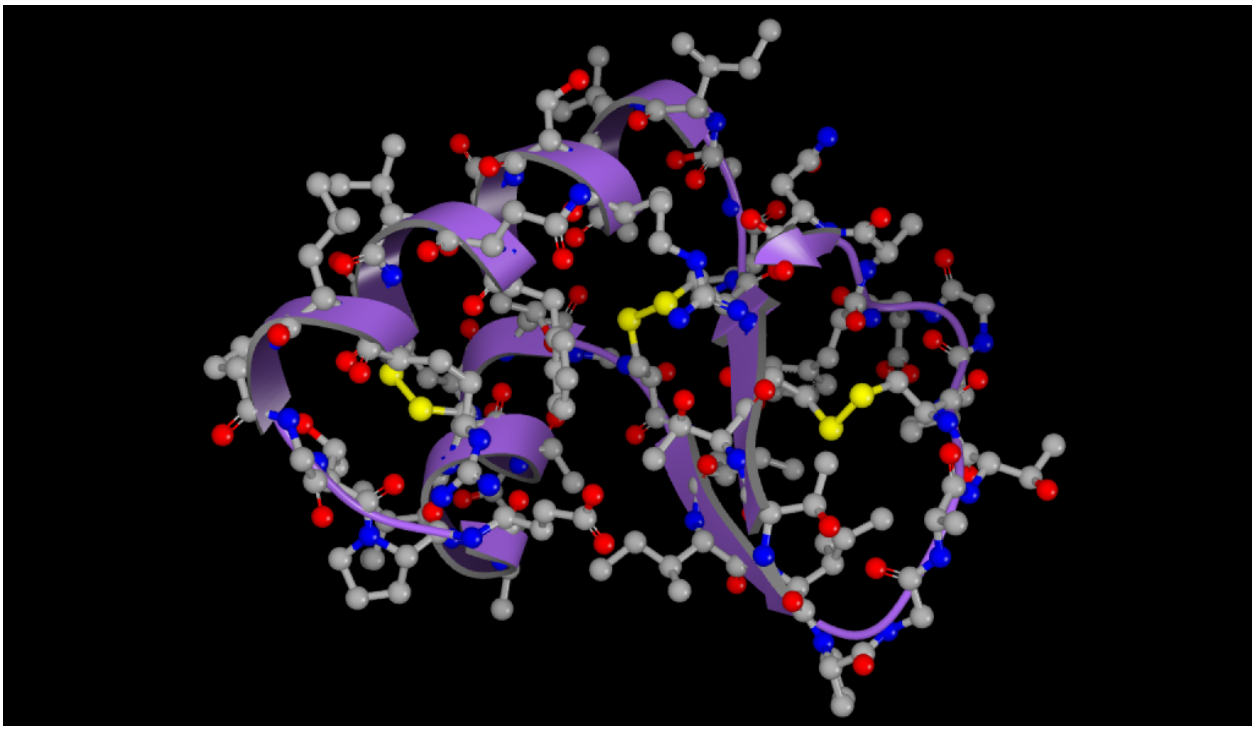
OEAtomStyle

The OEAtomStyle namespace encodes symbolic constants representing the style of the atoms in an *OE3DMolStyle*. The constants can be used as an argument for the `OE3DMolStyle.SetAtomStyle` function in *OE3DMolStyle*.

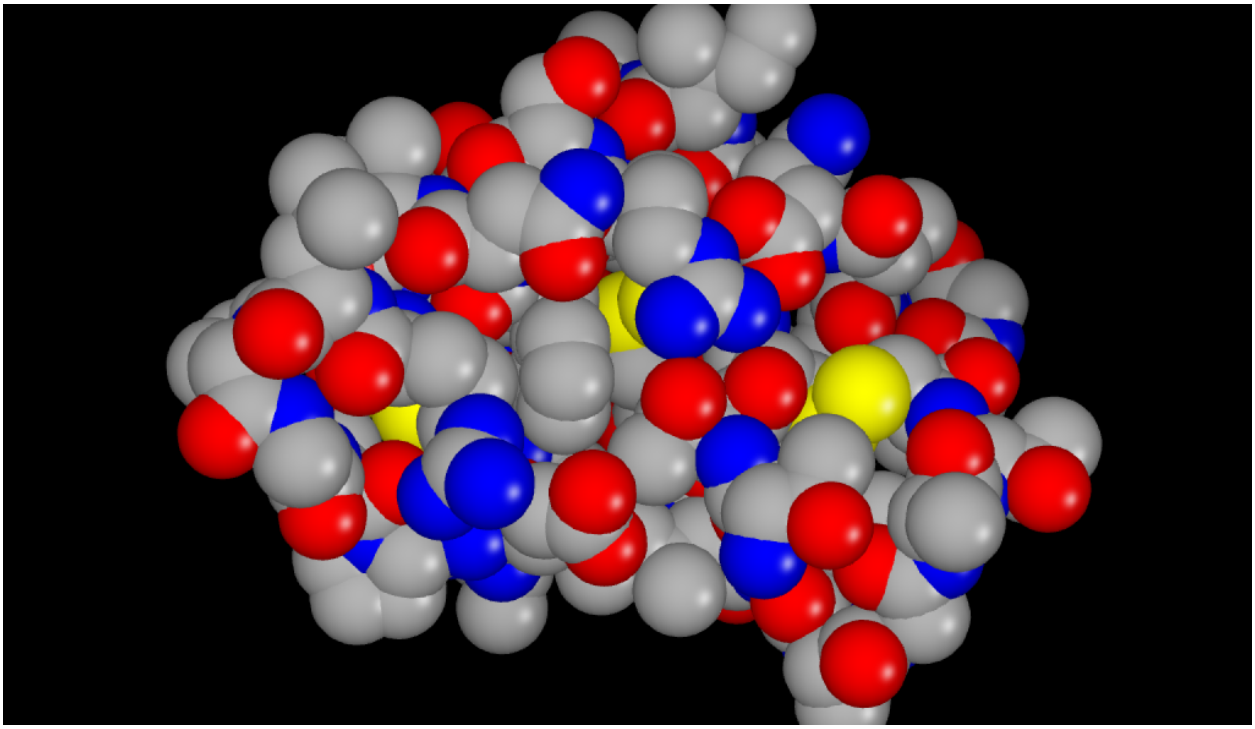
This namespace contains constants.

BEGIN

BallAndStick

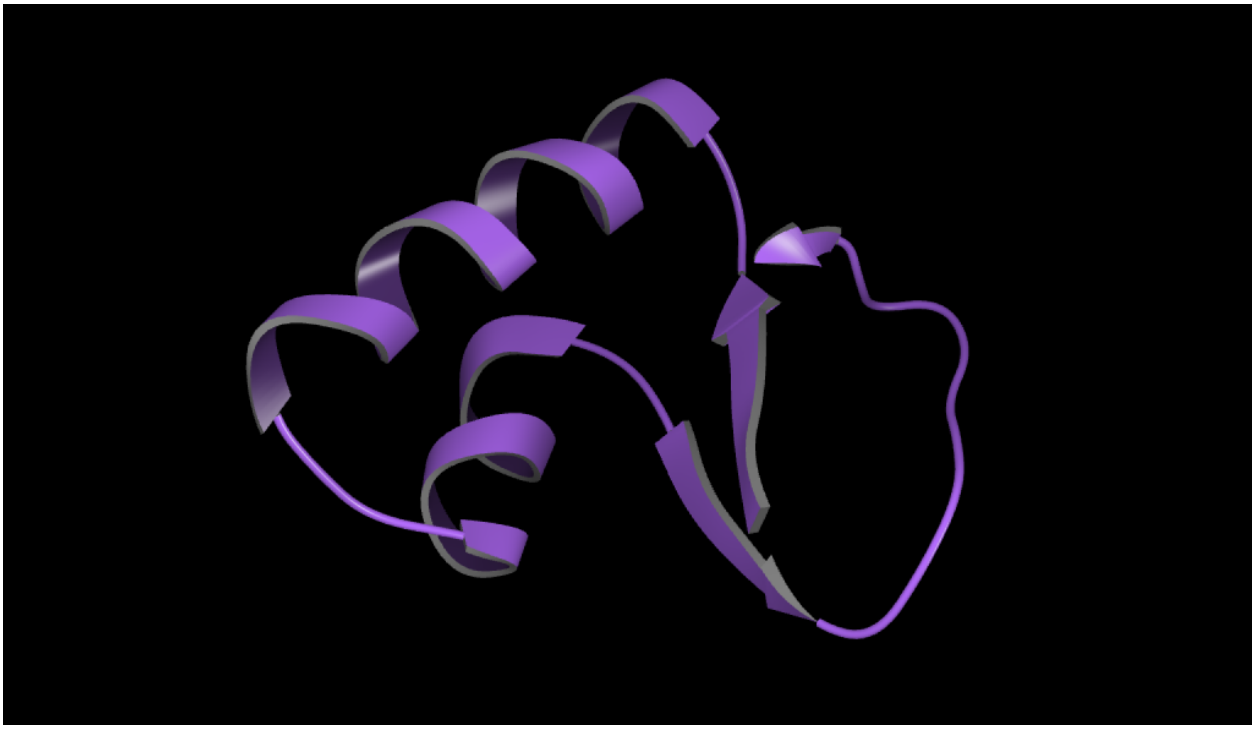


CPK



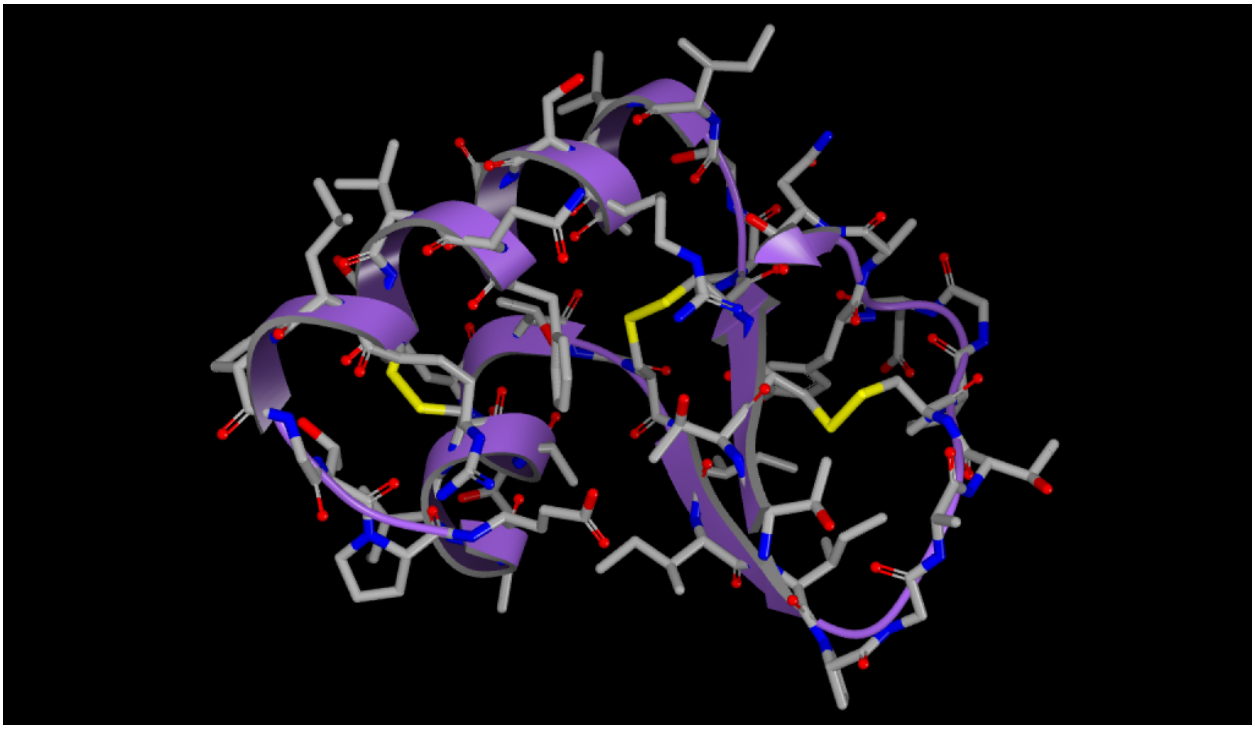
END

Hidden

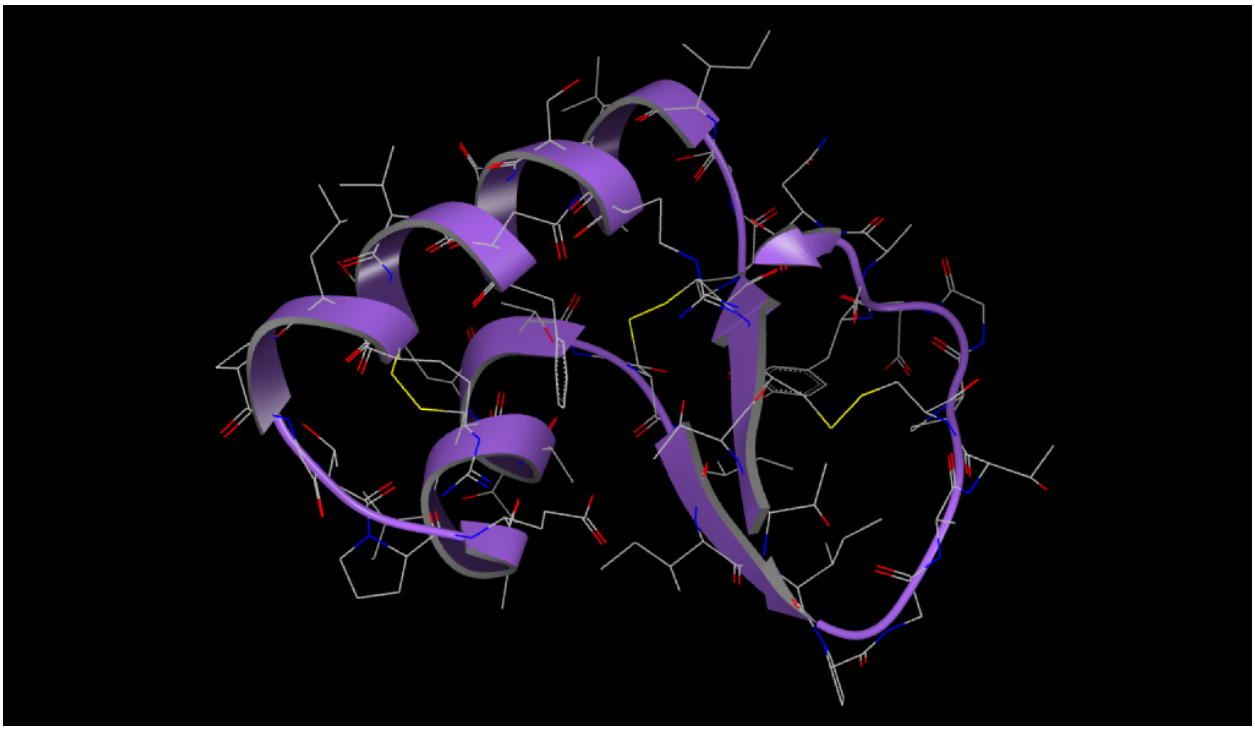


Stars

Stick



Wireframe



OEBaseColorScheme

The OEBaseColorScheme namespace encodes symbolic constants representing the base color scheme of the molecule associated with the *OE3DMolStyle*.

This namespace contains constants.

BEGIN

END

GenericData

Solid

OEBondStereo

See also

- stereo bond
- *OEBondBase.GetStereo* method
- *OEBondBase.SetStereo* method
- *OEBondBase.HasStereoSpecified* method

This namespace contains the following constants:

All

Represents all bond stereochemistry classes.

Cis

Bond stereochemistry value of the *OEBondStereo.CisTrans* class.

CisTrans

Bond stereochemistry class.

See also

OEBondBase.SetStereo function

DoubleEither

Bond stereochemistry value of the *OEBondStereo.CisTrans* class. Represents double bonds with unspecified cis or trans configuration (bow tie).

Hash

Bond type that can represent atom stereo information.

See also

OEMDLPerceiveBondStereo function

Trans

Bond stereochemistry value of the *OEBondStereo.CisTrans* class.

Wavy

Bond type that can represent atom stereo information.

See also

OEMDLPerceiveBondStereo function

Wedge

Bond type that can represent atom stereo information.

See also

OEMDLPerceiveBondStereo function

Undefined

Represents undefined bond stereochemistry.

OECIPAtomStereo

The *OECIPAtomStereo* namespace encodes symbolic constants representing Cahn–Ingold–Prelog atom stereo descriptors.

See also

- *OEPerceiveCIPStereo* function
- *OESetCIPStereo* function

This namespace contains the following constants:

NotStereo

Indicates atom that are not CIP stereo centers.

S

Indicates **S** CIP atom stereo centers.

R

Indicates **R** CIP atom stereo centers.

UnspecStereo

Indicates atoms that are CIP stereo centers but without specific stereo information (*i.e.* *OEAtomBase.HasStereoSpecified* method returns `false`)

OECIPBondStereo

The *OECIPAtomStereo* namespace encodes symbolic constants representing Cahn–Ingold–Prelog bond stereo descriptors.

See also

- *OEPerceiveCIPStereo* function
- *OESetCIPStereo* function

This namespace contains the following constants:

NotStereo

Indicates bonds that are not CIP stereo centers.

E

Indicates **E** CIP bond stereo centers.

Z

Indicates **Z** CIP bond stereo centers.

UnspecStereo

Indicates bonds that are CIP stereo centers but without specific stereo information (*i.e.* *OEBondBase.HasStereoSpecified* method returns `false`)

OEContourStyle

The OEContourStyle namespace encodes symbolic constants representing the contour style of the *OE3DMolStyle*. The constants can be used as an argument for the *OE3DMolStyle.SetContourStyle* function in *OE3DMolStyle*.

This namespace contains constants.

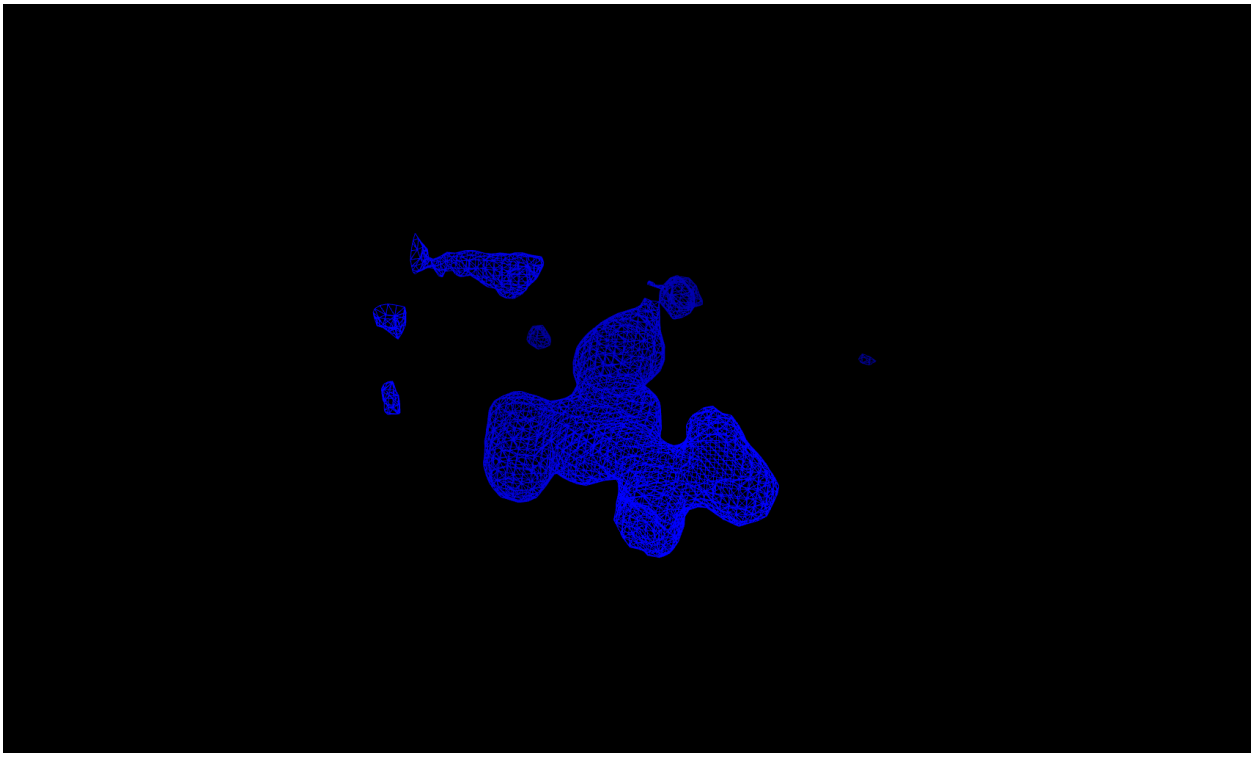
BEGIN

Cloud

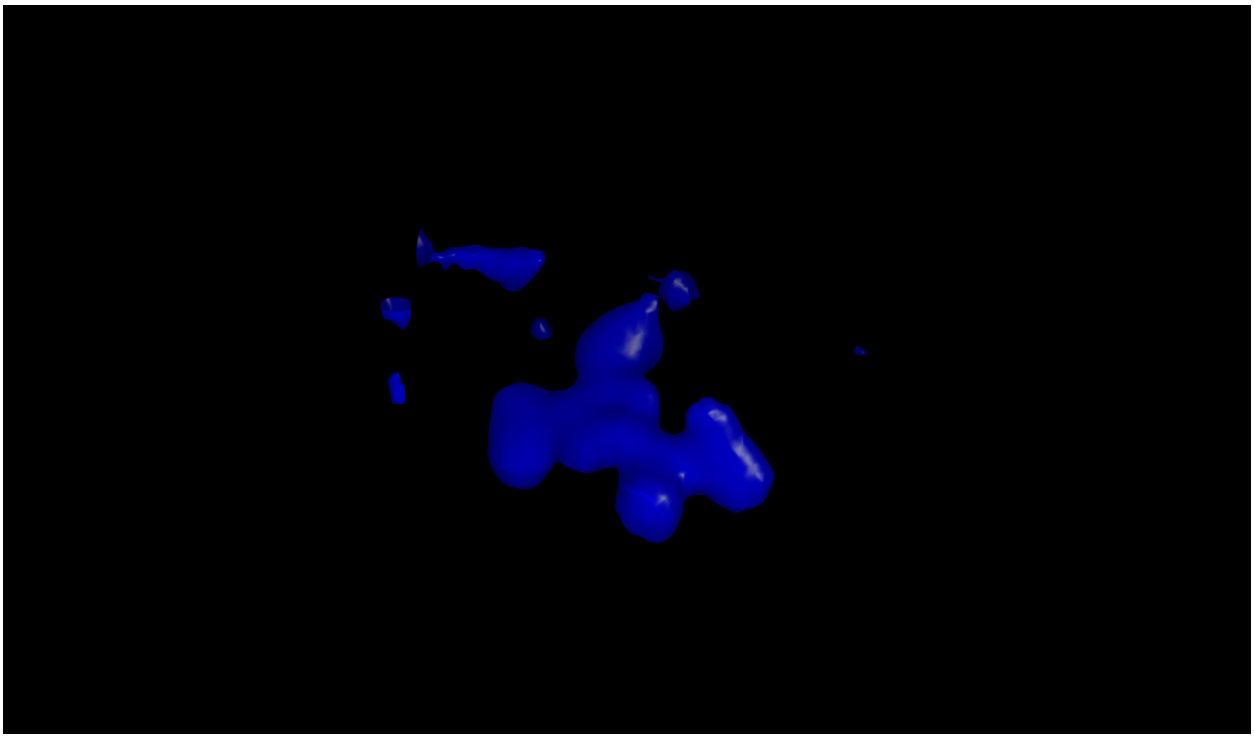
END

Hidden

Mesh



Solid



OECoordsType

This namespace contains constants used to interrogate the floating point precision of a given *OEConfBase.GetCoordsPtr* implementation.

DoubleType

FloatType

HalfFloatType

LongDoubleType

MaxType

Undefined

OEElemNo

The *OEElemNo* namespace encodes symbolic constants representing atomic numbers of the elements. Elements are represented by their atomic symbols encoding the integer value of their atomic number, *i.e.* *OEElemNo.C* has the value 6, *OEElemNo.O* the value 8 and *OEElemNo.H* the value 1.

This namespace contains the following constants:

H

Hydrogen

He

Helium

Li

Lithium

Be

Beryllium

B

Boron

C

Carbon

N

Nitrogen

O

Oxygen

F

Fluorine

Ne

Neon

Na

Sodium

Mg

Magnesium

Al

Aluminium

Si

Silicon

P

Phosphorus

S

Sulfur or sulphur

Cl

Chlorine

Ar

Argon

K

Potassium

Ca

Calcium

Sc

Scandium

Ti

Titanium

V

Vanadium

Cr

Chromium

Mn

Manganese

Fe

Iron

Co

Cobalt

Ni

Nickel

Cu

Copper

Zn

Zinc

Ga

Gallium

Ge

Germanium

As

Arsenic

Se

Selenium

Br

Bromine

Kr

Krypton

Rb

Rubidium

Sr

Strontium

Y

Yttrium

Zr

Zirconium

Nb

Niobium

Mo

Molybdenum

Tc

Technetium

Ru

Ruthenium

Rh

Rhodium

Pd

Palladium

Ag

Silver

Cd

Cadmium

In

Indium

Sn

Tin

Sb

Antimony

Te

Tellurium

I

Iodine

Xe

Xenon

Cs

Caesium or cesium

Ba

Barium

La

Lanthanum

Ce

Cerium

Pr

Praseodymium

Nd

Neodymium

Pm

Promethium

Sm

Samarium

Eu

Europium

Gd

Gadolinium

Tb

Terbium

Dy

Dysprosium

Ho

Holmium

Er

Erbium

Tm

Thulium

Yb

Ytterbium

Lu

Lutetium

Hf

Hafnium

Ta

Tantalum

W

Tungsten

Re

Rhenium

Os

Osmium

Ir

Iridium

Pt

Platinum

Au

Gold

Hg

Mercury

Tl

Thallium

Pb

Lead

Bi

Bismuth

Po

Polonium

At

Astatine

Rn

Radon

Fr

Francium

Ra

Radium

Ac

Actinium

Th

Thorium

Pa

Protactinium

U

Uranium

Np

Neptunium

Pu

Plutonium

Am

Americium

Cm

Curium

Bk

Berkelium

Cf

Californium

Es

Einsteinium

Fm

Fermium

Md

Mendelevium

No

Nobelium

Lr

Lawrencium

Rf

Rutherfordium

Db

Dubnium

Sg

Seaborgium

Bh

Bohrium

Hs

Hassium

Mt

Meitnerium

Ds

Darmstadtium

Rg

Roentgenium

Cn

Copernicium

Nh

Nihonium

Fl

Flerovium

Mc

Moscovium

Lv

Livermorium

Ts

Tennessine

Og

Oganesson

MAXELEM

Du

Dummy atom

Lp

Lone pair

Xx

D

Deuterium (^2H), hydrogen isotope that contains one proton and one neutron in its nucleus.

T

Tritium (^3H), hydrogen isotope that contains one proton and two neutrons in its nucleus.

OEExprOpts

Pattern matching in **OEChem TK** is always done using query molecules or query graphs (*OEQMolBase*). Non-query molecules, *i.e.* those that are defined by *OEMolBase* abstract base-class must be converted into a query molecule before using them in pattern matching algorithms (*OESubSearch*, *OEMCSSearch* or *OECliqueSearch*).

The `OEExprOpts` namespace contains expression options that control the conversion into query molecules.

See also

- *OESubSearch.Init* method
- *OEMCSSearch.Init* method
- *OEQMolBase.BuildExpressions* method

The *OEExprOpts* namespace contains both atom and bond expressions options. Some options such as *OEExprOpts.Aromaticity* can be used both as an atom or a bond expression.

Table 3: **Atom and bond expression options**

Constant name	Atom expression	Bond expression	Should be used with
Aromaticity	**X**	**X**	
AtomicNumber	**X**		
BondOrder		**X**	
Chiral	**X**	**X**	
Degree	**X**		
EqAromatic	**X**		*AtomicNumber*
EqCAliphaticONS	**X**		*AtomicNumber*
EqCHalogen	**X**		*AtomicNumber*
EqCPSAcidRoot	**X**		*AtomicNumber*
EqDoubleTriple		**X**	*BondOrder*
EqHalogen	**X**		*AtomicNumber*
EqKetoneSulfoneRoot	**X**		*AtomicNumber*
EqMetal	**X**		*AtomicNumber*
EqNotAromatic	**X**	**X**	*AtomicNumber* or *BondOrder*
EqON	**X**		*AtomicNumber*
EqONS	**X**		*AtomicNumber*
EqPS	**X**		*AtomicNumber*
EqSingleDouble		**X**	*BondOrder*
ExplicitDegree	**X**		
FormalCharge	**X**		
HCount	**X**		
HvyDegree	**X**		
Hybridization	**X**		
ImplicitHCount	**X**		
IntType	**X**	**X**	
Mass	**X**		
RingMember	**X**	**X**	
StrictFormalCharge	**X**		
StringType	**X**	**X**	
SymmetryClass	**X**	**X**	
Valence	**X**		

The *OEExprOpts* namespace also contains some predefined atom and bonds expression combinations listed in the table below.

Table 4: **Predefined atom and bond expression combinations**

Constant name	Combination of expression flags
AutomorphAto	*AtomicNumber* | *Aromaticity* | *RingMember* | *HvyDegree*
AutomorphBon	*Aromaticity*
DefaultAtoms	*AtomicNumber* | *Aromaticity* | *FormalCharge*
DefaultBonds	*BondOrder* | *Aromaticity*
ExactAtoms	*AtomicNumber* | *Aromaticity* | *StrictFormalCharge* | *Degree* | *HCount* | *Chiral* | *Mass* | *RingMember*
ExactBonds	*BondOrder* | *Aromaticity* | *Chiral* | *RingMember*

Aromaticity

The *OEExprOpts.Aromaticity* flag can be used both as atom and bond expression option.

- When using this option as an atom expression option, atoms are considered equivalent if the *OEAtomBase.IsAromatic* method returns the same value for them.
- When using this option as a bond expression option, bonds are considered equivalent if the *OEBondBase.IsAromatic* method returns the same value for them.

Table 5: **Example of using the *OEExprOpts.Aromaticity* option in substructure search**

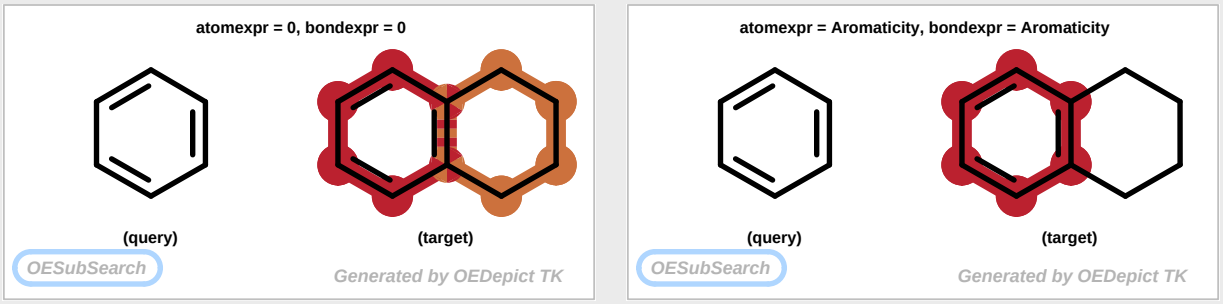
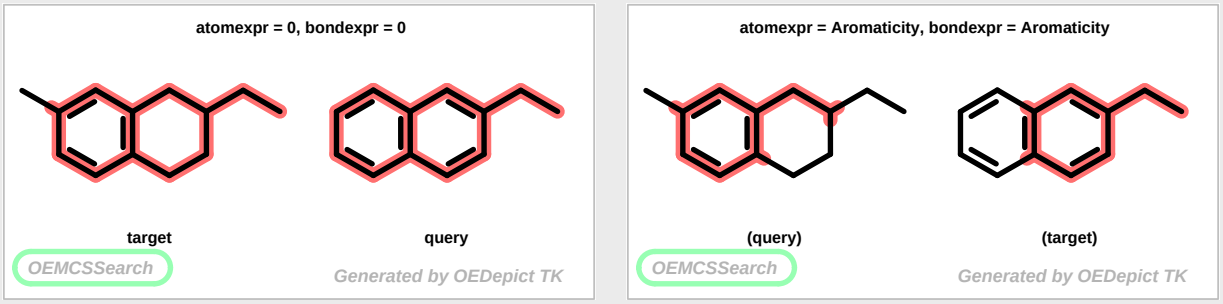


Table 6: **Example of using the *OEExprOpts.Aromaticity* option in maximum common substructure search**



AtomicNumber

The *OEExprOpts.AtomicNumber* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetAtomicNum* method returns the same value for them.

Table 7: **Example of using the *OEExprOpts.AtomicNumber* option in substructure search**

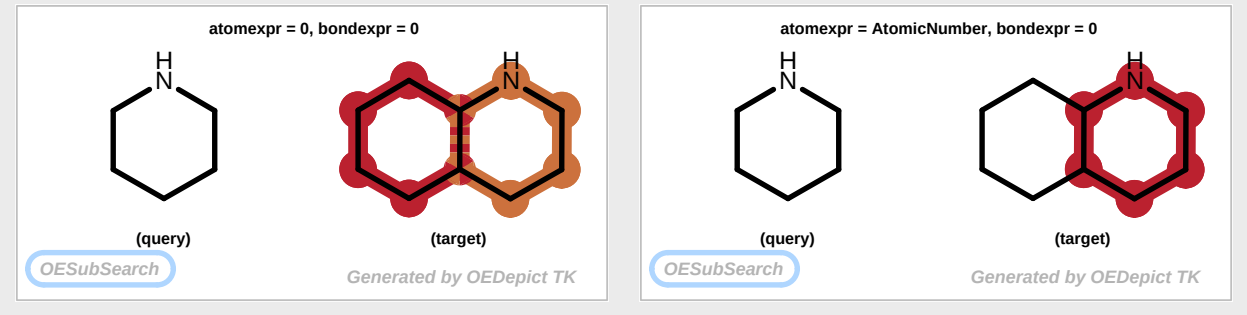
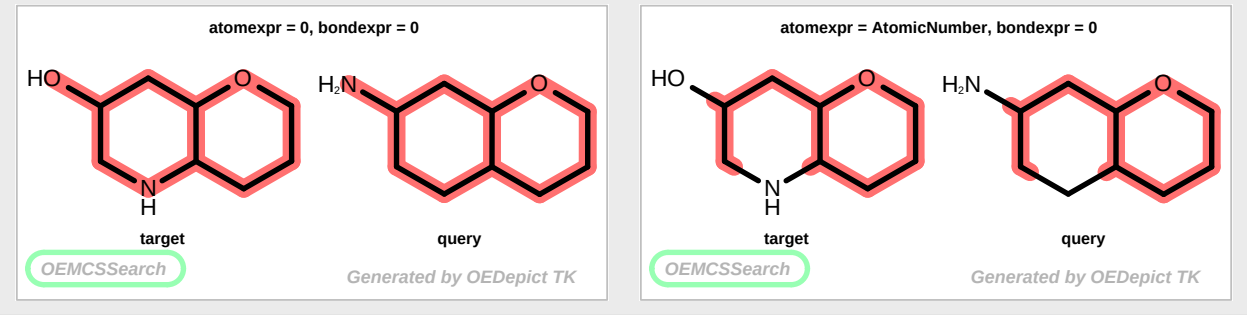


Table 8: **Example of using the *OEExprOpts.AtomicNumber* option in maximum common substructure search**



AutomorphAtoms

This constant along with *OEExprOpts.AutomorphBonds* is used when determining automorphism related graphs *OEExprOpts.AutomorphAtoms* is the combination of the following atom expression constants:

- *OEExprOpts.AtomicNumber*
- *OEExprOpts.Aromaticity*
- *OEExprOpts.RingMember*
- *OEExprOpts.HvyDegree*

AutomorphBonds

This constant along with *OEExprOpts.AutomorphAtoms* is used when determining automorphism related graphs. *OEExprOpts.AutomorphBonds* is equivalent to the *OEExprOpts.Aromaticity* constant.

BondOrder

The *OEExprOpts.BondOrder* flag can be used as a bond expression option. When using this option, bonds are considered equivalent if the *OEBondBase.GetOrder* method returns the same value for them.

Table 9: **Example of using the *OEExprOpts.BondOrder* option in substructure search**

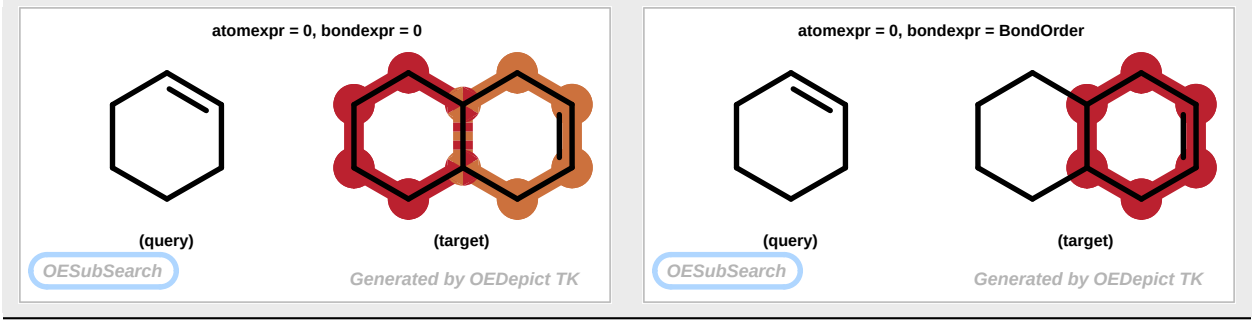
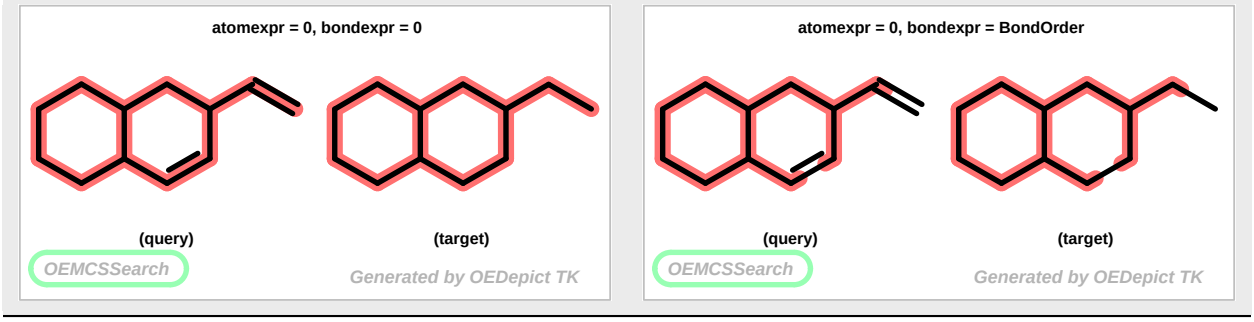


Table 10: **Example of using the *OEExprOpts.BondOrder* option in maximum common substructure search**



DefaultAtoms

This constant along with *OEExprOpts.DefaultBonds* is used when a moderate degree of discrimination is required of the graph equivalence. *OEExprOpts.DefaultAtoms* is the combination of the following atom expression constants:

- *OEExprOpts.AtomicNumber*
- *OEExprOpts.Aromaticity*
- *OEExprOpts.FormalCharge*

This combination of flags is intended to give a moderate degree of discrimination of the graph equivalence of atoms. Accordingly, two atoms are considered equivalent if they have the same element, aromaticity value and formal charge.

DefaultBonds

OEExprOpts.DefaultBonds is the combination of the following bond expression constants:

- *OEExprOpts.BondOrder*
- *OEExprOpts.Aromaticity*

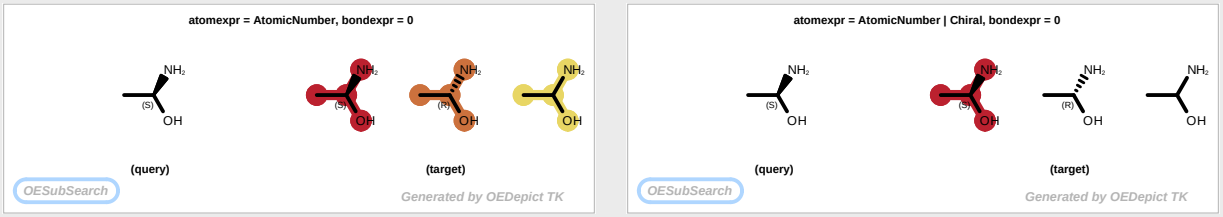
This combination of flags is intended to give a moderate degree of discrimination of the graph equivalence of bonds. Accordingly, two bonds are considered equivalent if they have the same bond order and aromaticity value.

Chiral

The *OEExprOpts.Chiral* flag can be used both as atom and bond expression option.

- When using this option as an atom expression option, if the query atom has specified tetrahedral configuration, than it can only be mapped to an atom with the same stereo configuration.

Table 11: **Example of using the *OEExprOpts.Chiral* atom option in substructure search**

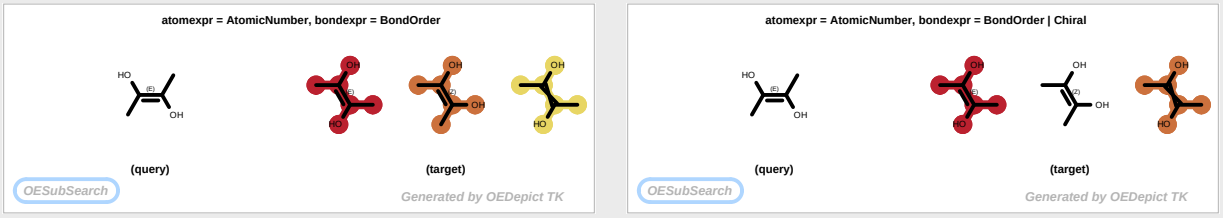


Warning

The *OEExprOpts.Chiral* is not available as an atom expression option in maximum common substructure search (*OEMCSSearch*) and clique search (*OECliqueSearch*).

- When using this option as a bond expression option, if the query bond has specified cis/trans configuration, than it can only be mapped to a bond with the same stereo configuration.

Table 12: **Example of using the *OEExprOpts.Chiral* atom option in substructure search**



Warning

The *OEExprOpts.Chiral* is not available as a bond expression option in maximum common substructure search (*OEMCSSearch*) and clique search (*OECliqueSearch*).

Degree

The *OEExprOpts.Degree* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetDegree* method returns the same value for them.

Hint

The *OEAtomBase.GetDegree* method returns the total degree including the contribution from implicit hydrogens.

EqAromatic

The *OEExprOpts.EqAromatic* flag can be used as an atom expression option. When using this option, aromatic atoms are considered equivalent regardless of their atomic number.

Warning

The *OEExprOpts.EqAromatic* should always be used with the *OEExprOpts.AtomicNumber* flag.

Table 13: **Example of using the *OEExprOpts.EqAromatic* option in substructure search**

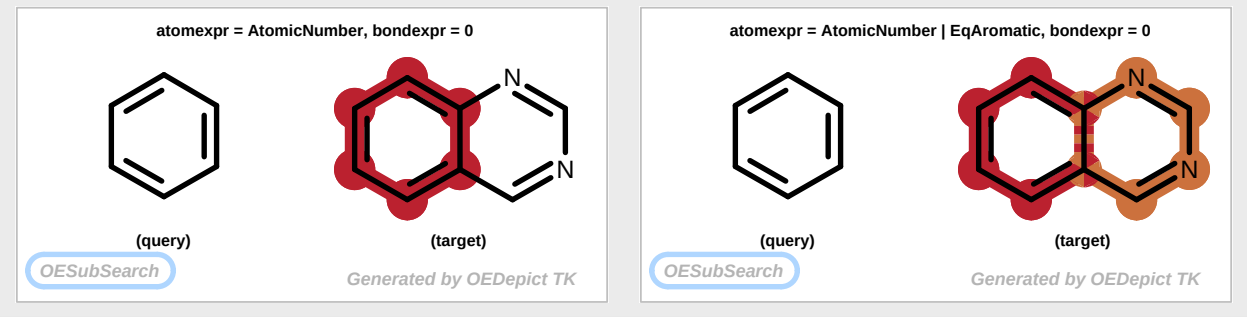
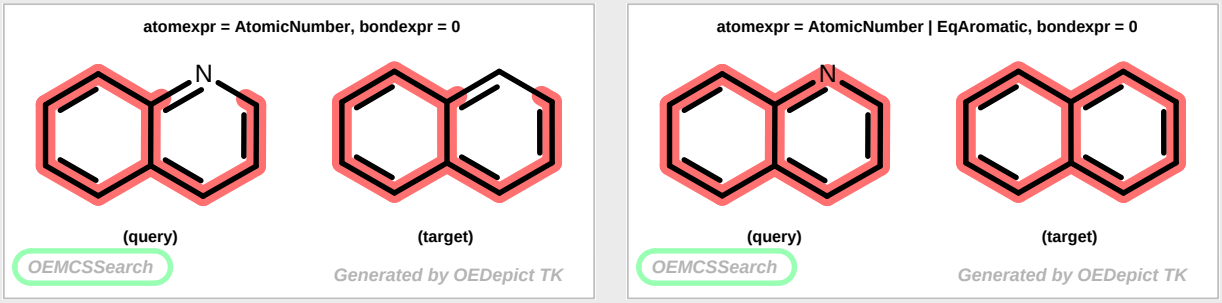


Table 14: **Example of using the *OEExprOpts.EqAromatic* option in maximum common substructure search**



EqCAliphaticONS

The *OEExprOpts.EqCAliphaticONS* flag can be used as an atom expression option. When using this option, aliphatic query carbon atoms are considered equivalent with aliphatic oxygen, nitrogen, or sulfur atoms.

Warning

The *OEExprOpts.EqCAliphaticONS* should always be used with the *OEExprOpts.AtomicNumber* flag.

Table 15: **Example of using the *OEExprOpts.EqCAliphaticONS* option in substructure search**

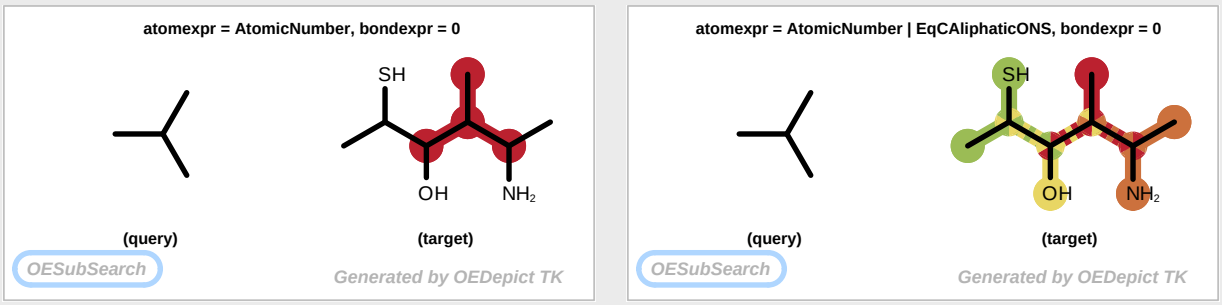
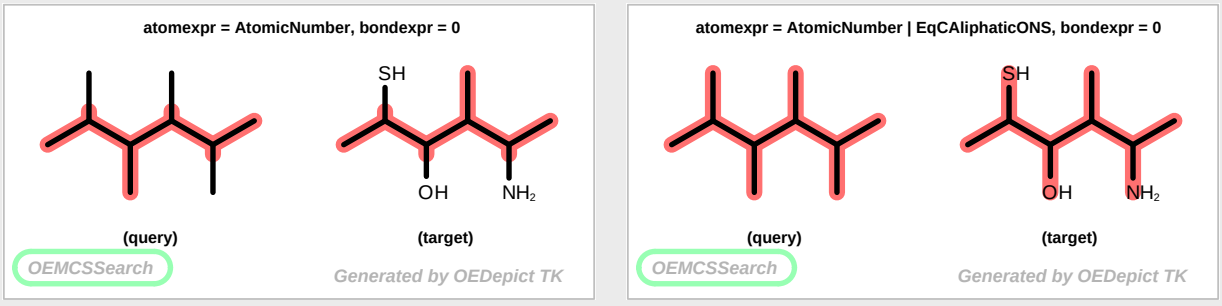


Table 16: **Example of using the *OEExprOpts.EqCAliphaticONS* option in maximum common substructure search**



EqCHalogen

The *OEExprOpts.EqCHalogen* flag can be used as an atom expression option. When using this option, carbons are considered equivalent with halogen atoms.

> **Warning**
>
> The *OEExprOpts.EqCHalogen* should always be used with the *OEExprOpts.AtomicNumber* flag.

EqCPSAcidRoot

The *OEExprOpts.EqCPSAcidRoot* flag can be used as an atom expression option. When using this option, query atoms which are the central atom of a carboxylic, phosphonic, or sulfonic acid moiety are equivalent to a potential match atom which is also the central atom of a carboxylic, phosphonic, or sulfonic moiety.

> **Warning**
>
> The *OEExprOpts.EqCPSAcidRoot* should always be used with the *OEExprOpts.AtomicNumber* flag.

EqDoubleTriple

The *OEExprOpts.EqDoubleTriple* flag can be used as a bond expression option. When using this option, double and triple bonds are considered equivalent.

> **Warning**
>
> The *OEExprOpts.EqDoubleTriple* should always be used with the *OEExprOpts.BondOrder* flag.

Table 17: **Example of using the *OEExprOpts.EqDoubleTriple* option in substructure search**

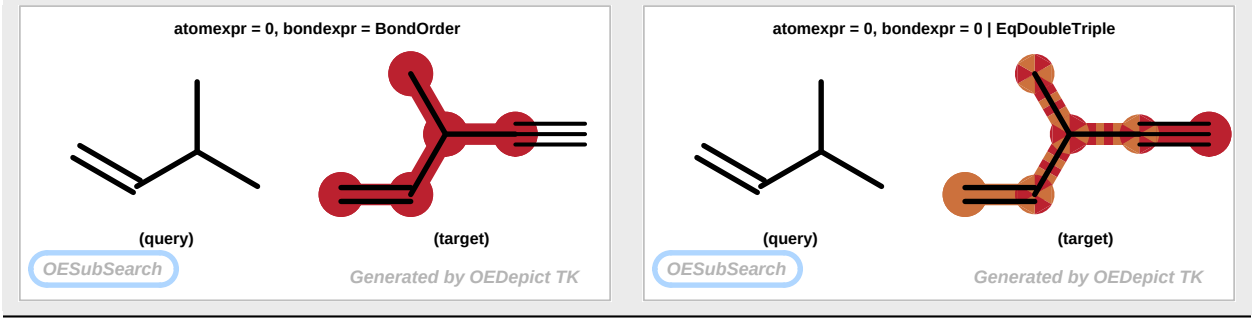
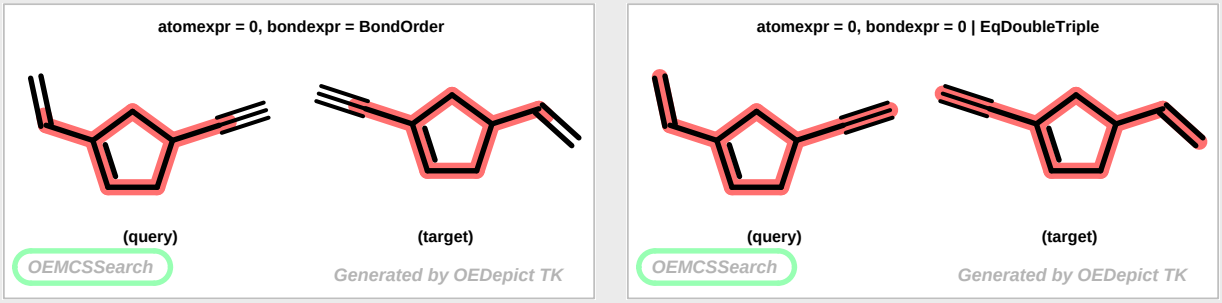


Table 18: **Example of using the *OEExprOpts.EqDoubleTriple* option in maximum common substructure search**



EqHalogen

The *OEExprOpts.EqHalogen* flag can be used as an atom expression option. When using this option, halide atoms (*OEElemNo.F*, *OEElemNo.Cl*, *OEElemNo.Br*, and *OEElemNo.I*) are considered equivalent.

> **Warning**
>
> The *OEExprOpts.EqHalogen* should always be used with the *OEExprOpts.AtomicNumber* flag.

Table 19: **Example of using the *OEExprOpts.EqHalogen* option in substructure search**

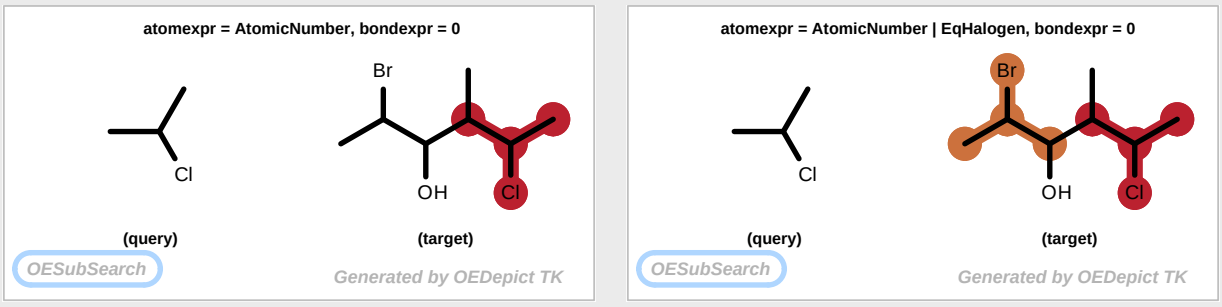
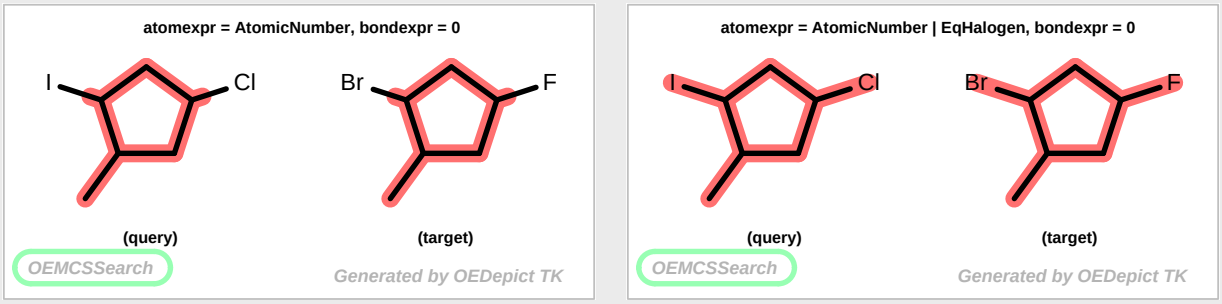


Table 20: **Example of using the *OEExprOpts.EqHalogen* option in maximum common substructure search**



EqKetoneSulfoneRoot

The *OEExprOpts.EqKetoneSulfoneRoot* flag can be used as an atom expression option. When using this option, query atoms which are the central atom of a ketone or sulfone moiety are equivalent to a potential match atom which is also the central atom of a ketone or sulfone moiety.

> **Warning**
>
> The *OEExprOpts.EqKetoneSulfoneRoot* should always be used with the *OEExprOpts.AtomicNumber* flag.

EqMetal

The *OEExprOpts.EqMetal* flag can be used as an atom expression option. When using this option, metal atoms are considered equivalent.

> **Warning**
>
> The *OEExprOpts.EqMetal* should always be used with the *OEExprOpts.AtomicNumber* flag.

EqNotAromatic

The *OEExprOpts.EqNotAromatic* flag can be used both as atom and bond expression option.

- When using this option as an atom expression option, aliphatic atoms are considered equivalent regardless of their atomic number.

Table 21: **Example of using the *OEExprOpts.EqNotAromatic* atom option in substructure search**

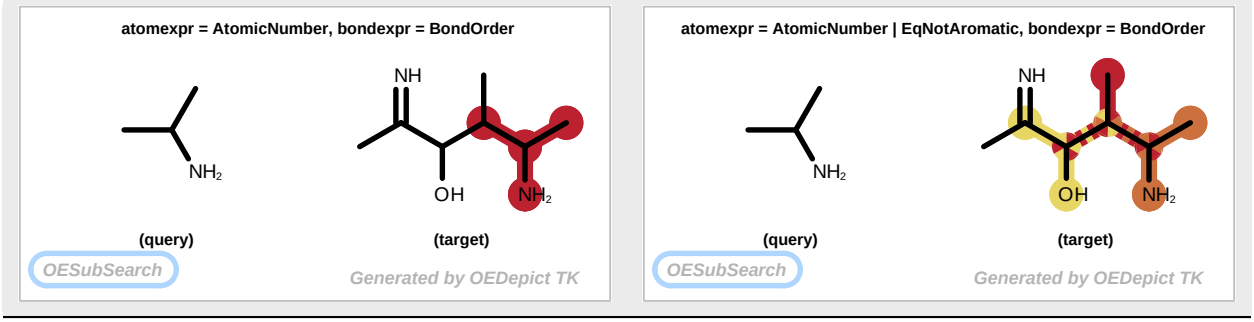
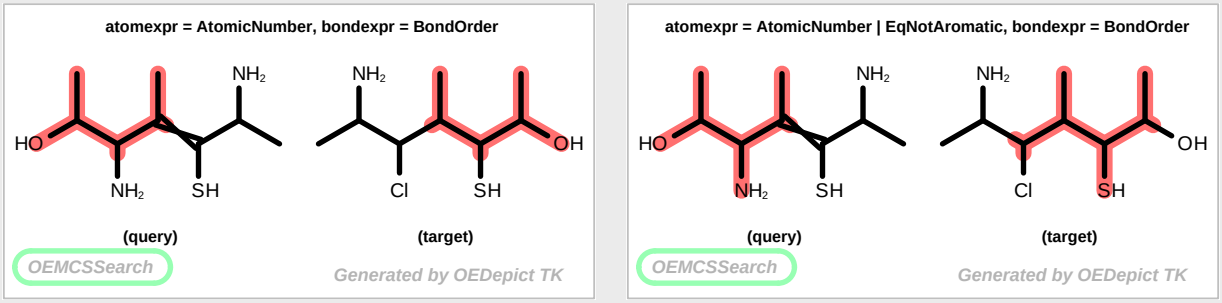


Table 22: **Example of using the *OEExprOpts.EqNotAromatic* atom option in maximum common substructure search**



- When using this option as a bond expression option, aliphatic bonds are considered equivalent regardless of their bond order.

Table 23: **Example of using the *OEExprOpts.EqNotAromatic* bond option in substructure search**

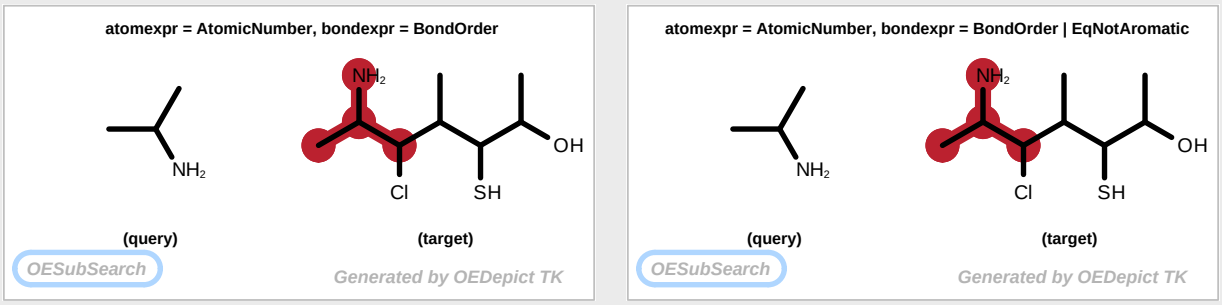
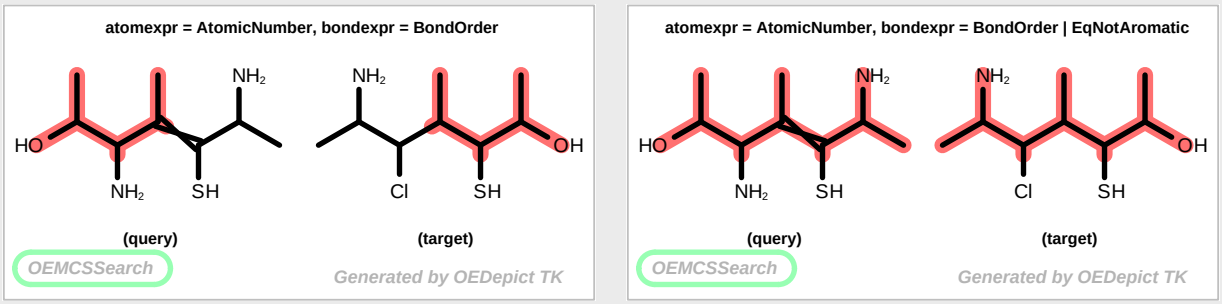


Table 24: **Example of using the *OEExprOpts.EqNotAromatic* bond option in maximum common substructure search**



Warning

The *OEExprOpts.EqNotAromatic* should always be used with either the *OEExprOpts.AtomicNumber* or the *OEExprOpts.BondOrder* flag.

EqON

The *OEExprOpts.EqON* flag can be used as an atom expression option. When using this option, nitrogen and oxygen atoms are considered equivalent.

> **Warning**
>
> The *OEExprOpts.EqON* should always be used with the *OEExprOpts.AtomicNumber* flag.

EqONS

The *OEExprOpts.EqONS* flag can be used as an atom expression option. When using this option, nitrogen, oxygen and sulfur atoms are considered equivalent.

> **Warning**
>
> The *OEExprOpts.EqONS* should always be used with the *OEExprOpts.AtomicNumber* flag.

EqPS

The *OEExprOpts.EqPS* flag can be used as an atom expression option. When using this option, phosphorus and sulfur atoms are considered equivalent.

> **Warning**
>
> The *OEExprOpts.EqPS* should always be used with the *OEExprOpts.AtomicNumber* flag.

EqSingleDouble

The *OEExprOpts.EqSingleDouble* flag can be used as a bond expression option. When using this option, single and double bonds and are considered equivalent.

Warning

The *OEExprOpts.EqSingleDouble* should always be used with the *OEExprOpts.BondOrder* flag.

Table 25: **Example of using the *OEExprOpts.EqSingleDouble* option in substructure search**

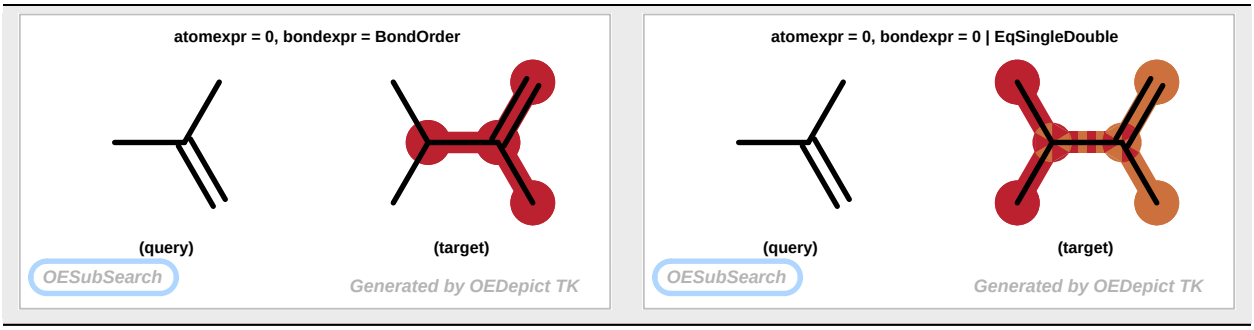
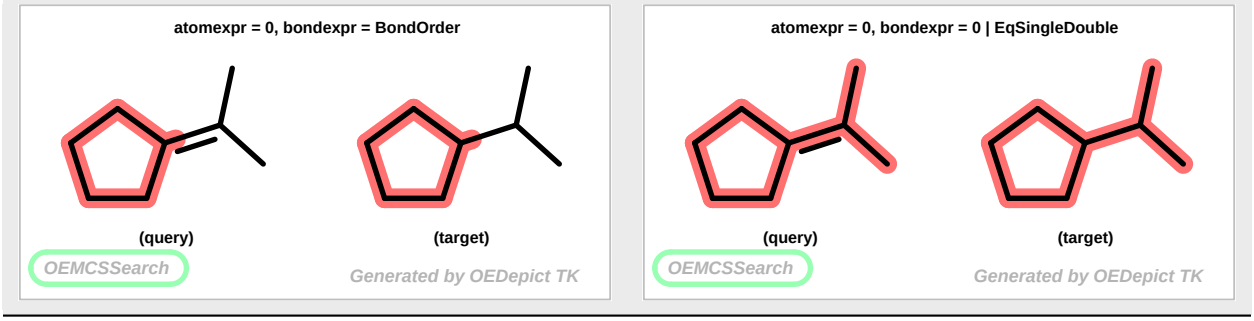


Table 26: **Example of using the *OEExprOpts.EqSingleDouble* option in maximum common substructure search**



ExactAtoms

OEExprOpts.ExactAtoms is the combination of the following bond expression constants:

- *OEExprOpts.AtomicNumber*
- *OEExprOpts.Aromaticity*
- *OEExprOpts.StrictFormalCharge*
- *OEExprOpts.Degree*
- *OEExprOpts.HCount*
- *OEExprOpts.Chiral*
- *OEExprOpts.Mass*
- *OEExprOpts.RingMember*

This combination of flags is intended to give a high degree of discrimination of the graph equivalence of atoms.

ExactBonds

OEExprOpts.ExactBonds is the combination of the following bond expression constants:

- *OEExprOpts.BondOrder*
- *OEExprOpts.Aromaticity*
- *OEExprOpts.Chiral*
- *OEExprOpts.RingMember*

This combination of flags is intended to give a high degree of discrimination of the graph equivalence of bonds.

ExplicitDegree

The *OEExprOpts.ExplicitDegree* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetHvyDegree* method returns the same value for them.

FormalCharge

The *OEExprOpts.FormalCharge* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetFormalCharge* method returns the same value for them **and** the query atom has a non-zero charge. If you want to consider zero charges see option *OEExprOpts.StrictFormalCharge*.

Table 27: **Example of using the *OEExprOpts.FormalCharge* option in substructure search**

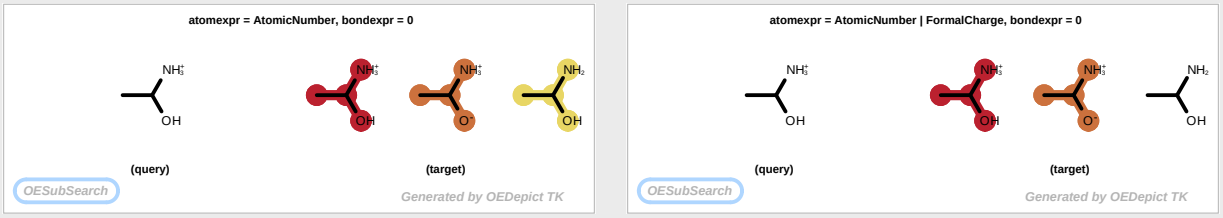
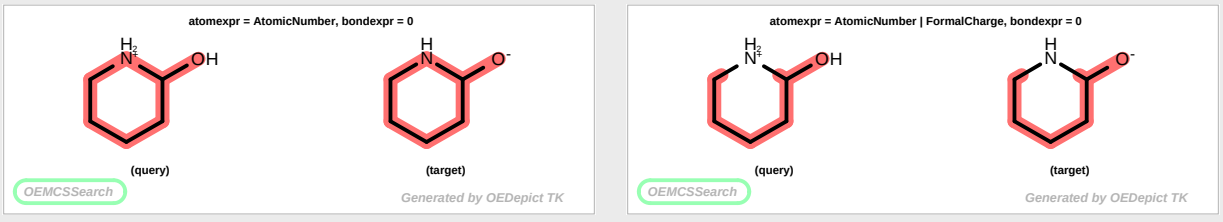


Table 28: **Example of using the *OEExprOpts.FormalCharge* option in maximum common substructure search**



See also

- *OEExprOpts.StrictFormalCharge* constants

HCount

The *OEExprOpts.HCount* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetTotalHCount* method returns the same value for them.

> **Hint**
>
> The *OEAtomBase.GetTotalHCount* method returns the number of explicit and implicit hydrogens i.e. the expression built when the *OEExprOpts.HCount* flag is set will therefore not discriminate between implicit and explicit hydrogen counts.

HvyDegree

The *OEExprOpts.HvyDegree* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetHvyDegree* method returns the same value for them.

Hybridization

The *OEExprOpts.Hybridization* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetHyb* method returns the same value for them.

> **Warning**
>
> Hybridization of the molecules must be perceived by calling the *OEAssignHybridization* function.

ImplicitHCount

The *OEExprOpts.ImplicitHCount* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetImplicitHCount* method returns the same value for them.

IntType

The *OEExprOpts.IntType* flag can be used both as atom and bond expression option.

- When using this option as an atom expression option, atoms are considered equivalent if the *OEAtomBase.GetIntType* method returns the same value for them.
- When using this option as a bond expression option, bonds are considered equivalent if the *OEBondBase.GetIntType* method returns the same value for them.

Mass

The *OEExprOpts.Mass* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetIsotope* method returns the same value for them.

RingMember

The *OEExprOpts.RingMember* flag can be used both as atom and bond expression option.

- When using this option as an atom expression option, atoms are considered equivalent if the *OEAtomBase.IsInRing* method returns the same value for them.

Table 29: **Example of using the *OEExprOpts.RingMember* atom option in substructure search**

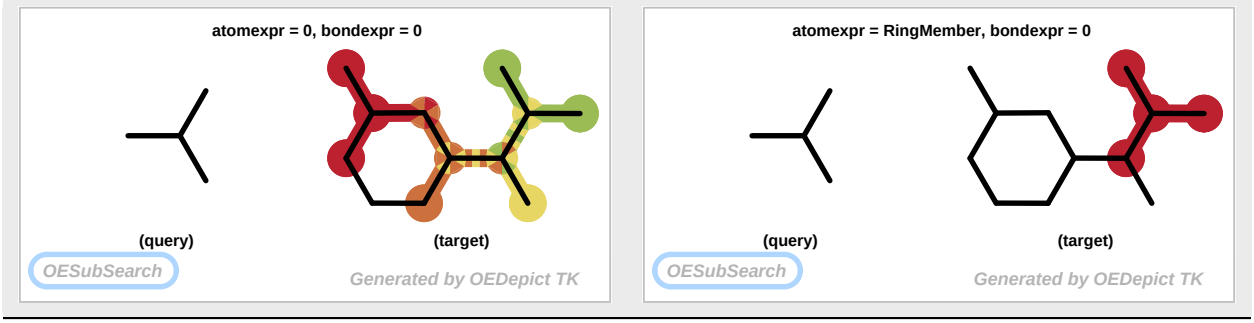
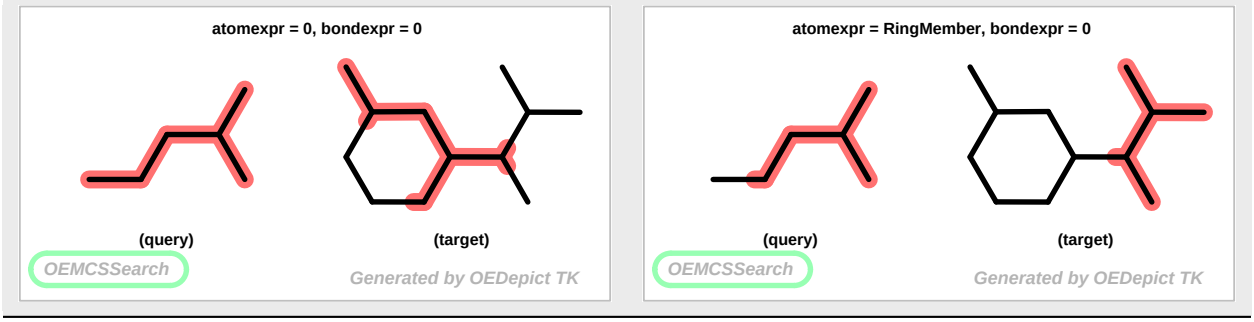


Table 30: **Example of using the *OEExprOpts.RingMember* atom option in maximum common substructure search**



- When using this option as a bond expression option, bonds are considered equivalent if the *OEBondBase.IsInRing* method returns the same value for them.

Table 31: **Example of using the *OEExprOpts.RingMember* bond option in substructure search**

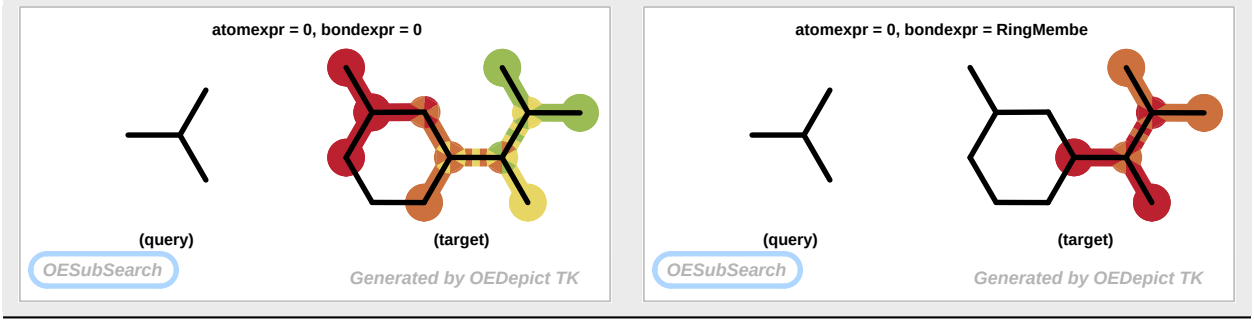
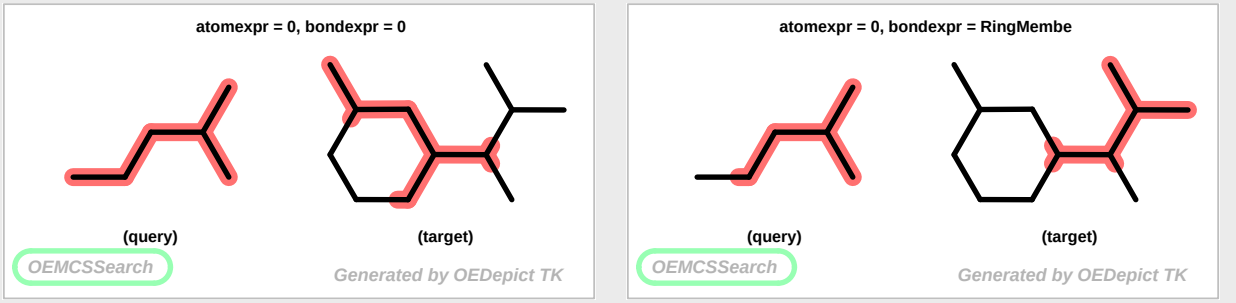


Table 32: **Example of using the *OEExprOpts.RingMember* bond option in maximum common substructure search**



StrictFormalCharge

The *OEExprOpts.StrictFormalCharge* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetFormalCharge* method returns the same value for them. The formal charges must be equivalent even if the formal charge of the query atom is zero.

Table 33: **Example of using the *OEExprOpts.StrictFormalCharge* option in substructure search**

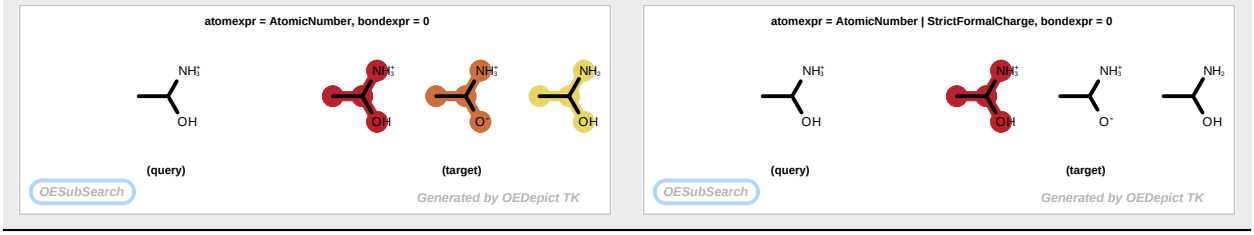
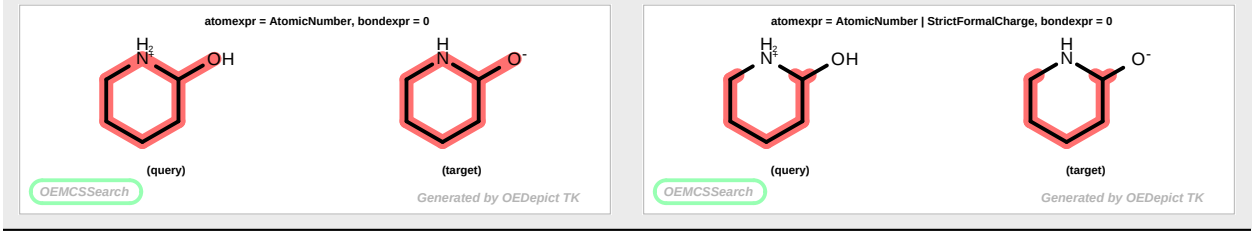


Table 34: **Example of using the *OEExprOpts.StrictFormalCharge* option in maximum common substructure search**



See also

- *OEExprOpts.FormalCharge* constant

StringType

The *OEExprOpts.StringType* flag can be used both as atom and bond expression option.

- When using this option as an atom expression option, atoms are considered equivalent if the *OEAtomBase.GetType* method returns the same value for them.
- When using this option as a bond expression option, bonds are considered equivalent if the *OEBondBase.GetType* method returns the same value for them.

SymmetryClass

The *OEExprOpts.SymmetryClass* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetSymmetryClass* method returns the same value for them.

Warning

Symmetry of the molecules must be perceived by calling the *OEPerceiveSymmetry* function.

Valence

The *OEExprOpts.Valence* flag can be used as an atom expression option. When using this option, atoms are considered equivalent if the *OEAtomBase.GetValence* method returns the same value for them.

Hint

The *OEAtomBase.GetValence* method includes the valence contribution from implicit hydrogens, therefore, the expression test may return successfully even when the explicit valence of two corresponding atoms differ.

OEExprType

This namespace contains constants.

AllowedElements

And

Aromatic

AromBondOrder

Utilized when optimizing bond expression trees.

AromElem

AromElemMinDegree

Utilized when optimizing atom expression trees.

AromOrBondOrder

Utilized when optimizing bond expression trees.

AromRingElem

Utilized when optimizing atom expression trees.

AtomParityChange

BondOrder

BondParityChange

Charge

Chiral

Connect

Const

DefaultEdge

Degree

DoubleAromBond

ElemCharge

Utilized when optimizing atom expression trees.

Element

ElemMinDegree

Utilized when optimizing atom expression trees.

ElemMinHCount

Utilized when optimizing bond expression trees.

FormalChargeChange

Halogen

HCount

HCountChange

Hyb

Implicit

IntType

Mass

Metal

MinDegree

MinHCount

Not

Or

Recurs

Ring

RingBondCount

RingBondOrder

Utilized when optimizing bond expression trees.

RingDefaultEdge

Utilized when optimizing bond expression trees.

RingElem

Utilized when optimizing atom expression trees.

Saturation

SingleAromBond

Size

Stereo

StringType

SymmetryClass

TerminalAtom

Undefined

Valence

OEFileStringType

This namespace contains constants that are used with *OESimpleAppOptions* and *OERefInputAppOptions* to control the input and output file parameters.

Mol

Any molecule file type.

Mol3D

Molecule fle types that contain 3D-coordinates.

MolReceptor

Molecule file types for receptor molecules.

BroodQMol

Molecule file type for Brood query.

DU

Design unit file type.

OEFileStringTypeFlavor

This namespace contains constants that are used with *OESimpleAppOptions* and *OERefInputAppOptions* to control the flavor of *OEFileStringType*.

Default

DEFAULT

The combination of *OEFileStringTypeFlavor.Pretty*, *OEFileStringTypeFlavor.FileList*, *OEFileStringTypeFlavor.Input*, *OEFileStringTypeFlavor.Output*, *OEFileStringTypeFlavor.MolOnly*. Generally useful for setting defaults, then disabling either *OEFileStringTypeFlavor.Input*, or *OEFileStringTypeFlavor.Output*,

Dim2D

Explicitly indicates 2D file types. See *OEIs2DFormat*.

Dim2D3D

Explicitly indicates 2D or 3D file types, and excludes 0D file types, valid for input types only. The combination of *OEFileStringTypeFlavor.Dim2D*, *OEFileStringTypeFlavor.Dim3D*,

Dim3D

Explicitly indicates 3D file types. See *OEIs3DFormat*.

FileList

Enables input file lists as valid filetypes, namely `.lst` and `.list`.

Pretty

Removes file types that are supported but not suggested for use.

Input

Explicitly specifies input file types. See *OEIsReadable*.

InputDefault

The combination of *OEFileStringTypeFlavor.FileList*, *OEFileStringTypeFlavor.Input*, *OEFileStringTypeFlavor.MolOnly*.

MolOnly

Explicitly indicates only molecule input file types, excludes reactions.

NoListDefault

The combination of *OEFileStringTypeFlavor.Input*, *OEFileStringTypeFlavor.Output*, *OEFileStringTypeFlavor.MolOnly*. Generally useful for setting defaults, then disabling either *OEFileStringTypeFlavor.Input*, or *OEFileStringTypeFlavor.Output*,

Output

Explicitly specifies output file types. See *OEIsWriteable*.

OutputDefault

The combination of *OEFileStringTypeFlavor.Output*, *OEFileStringTypeFlavor.MolOnly*.

RxnOnly

Explicitly indicates only reaction input file types.

OEFormat

The OEFormat namespace encodes symbolic constants representing the molecular file formats that may be read or written by **OEChem TK**.

See also

- *File Formats* section

This namespace contains the following constants:

Default

The default file format type is *OEFormat.SMI*.

CAN

This constant indicates Canonical SMILES file format.

CDX

This constant indicates ChemDraw CDX file format.

CIF

This constant indicates the small molecule Crystallographic Information File (CIF) format.

CSV

This constant indicates the comma separated values file format as specified by RFC 4180. The first column is canonical isomeric SMILES, the second column is the *molecule title*, the remaining columns are SD data as determined by the first molecule read or written to the *oemolstreambase*.

See also

CSV File Format

CXSMILES

This constant indicates the Chemaxon Extended SMILES format.

Enhanced stereogroup information is the only information in cxsmiles appendices currently supported. Please contact support@eyesopen.com if there are additional cxsmiles features that you would like to see supported.

The specific features supported for this format are limited to, MDL Chiral OFF and Enhanced stereochemical representation . The supported features cannot be in the presence of other unsupported features since a partial parse of the appendix information is disallowed.

See Chemaxon Extended SMILES and SMARTS for a complete specification of all possible features.

FASTA

This constant indicates FASTA protein sequence file format.

INCHI

This constant indicates IUPAC InChI file format.

INCHIKEY

This constant indicates IUPAC InChI Key file format.

ISM

This constant indicates Canonical isomeric SMILES file format. This is the same as the *OEFormat.SMI* file format.

MAXFORMAT

The maximum integral value any constant in the *OEFormat* can have.

> **Note**
>
> Values in this namespace are guaranteed to be contiguous.

MDL

This constant indicates MDL Mol file format.

MF

This constant indicates Molecular Formula (Hill order) format.

MMCIF

This constant indicates macromolecular Crystallographic Information File (mmCIF) file format.

MMOD

This constant indicates Macromodel file format.

MOL2

This constant indicates Tripos Sybyl mol2 file format.

MOL2H

This constant indicates Tripos Sybyl mol2 file format with explicit hydrogens.

MOPAC

This constant indicates MOPAC file format.

OEB

This constant indicates **OpenEye** OEBinary file format.

OEZ

This constant indicates Zstd compressed **OpenEye** OEBinary file format. Currently this file format **only** supports multi-conformer molecules derived from the *OEMCMolBase* class.

Note

This file format was introduced in the `2019.Apr` toolkits. OpenEye applications or toolkits released before Apr/2019 will **not** be able to read this new file format.

See also

- *Compressed Input and Output* section

PDB

This constant indicates Protein Databank PDB file format.

RDF

This constant indicates MDL RDF reaction file format.

SDF

This constant indicates MDL SD file format.

SKC

This constant indicates MDL ISIS Sketch file format.

SLN

This constant indicates Tripos Sybyl Line Notation file format.

SMI

This constant indicates canonical isomeric SMILES file format. This is the same as the *OEFormat.ISM* file format.

UNDEFINED

This constant indicates an unrecognized or undefined file format.

USM

This constant indicates SMILES file format.

XYZ

This constant indicates XMol XYZ file format.

OEFuzzVal

The *OEFuzzVal* namespace encodes symbolic constants representing fuzzy values used to construct *OEFuzzy* objects.

This namespace contains the following constants:

False

This constant represents fuzzy value of false.

True

This constant represents fuzzy value of true.

Maybe

This constant represents fuzzy value of maybe.

OEGroupType

The *OEGroupType* namespace encodes symbolic constants representing various group ids.

Component

This tag identifies a set atoms that were identified as originating from the same reactant or product component. This information allows multi-fragment reaction components to be identified.

MDLAbsStereo

This tag identifies the set of stereo centers with absolute stereochemistry.

MDLOrStereo

This tag identifies the set of stereo centers with relative (to other centers in the set) stereochemistry. The actual stereochemistry could be exactly as marked, or with all centers of the opposite parity. Only one of the two configurations is assumed to be present.

MDLAndStereo

This tag identifies the set of stereo centers with racemic stereochemistry. A mixture of the two enantiomeric configurations is present, one with the stereo centers of the marked parity, and the other with all the stereo centers of the opposite parity. No information about the relative amount of one or the other enantiomer is assumed or implied.

Monomer

This tag identifies the set of atoms that are belong to a specific monomer as parsed when a molecule is initialized with *OEHelmToMol*.

Undefined

UserDefined

This tag identifies a set atoms and/or bonds that were applied by an explicit user operation. Groups defined with this tag are supported for serialization in the `.oeb` format.

OEHybridization

The *OEHybridization* namespace encodes symbolic constants representing atom hybridizations.

See also

- *OEAssignHybridization* function
- *OEAtomBase.GetHyb* method
- *OEAtomBase.SetHyb* method

This namespace contains the following constants:

Unknown

sp

sp2

sp3

sp3d

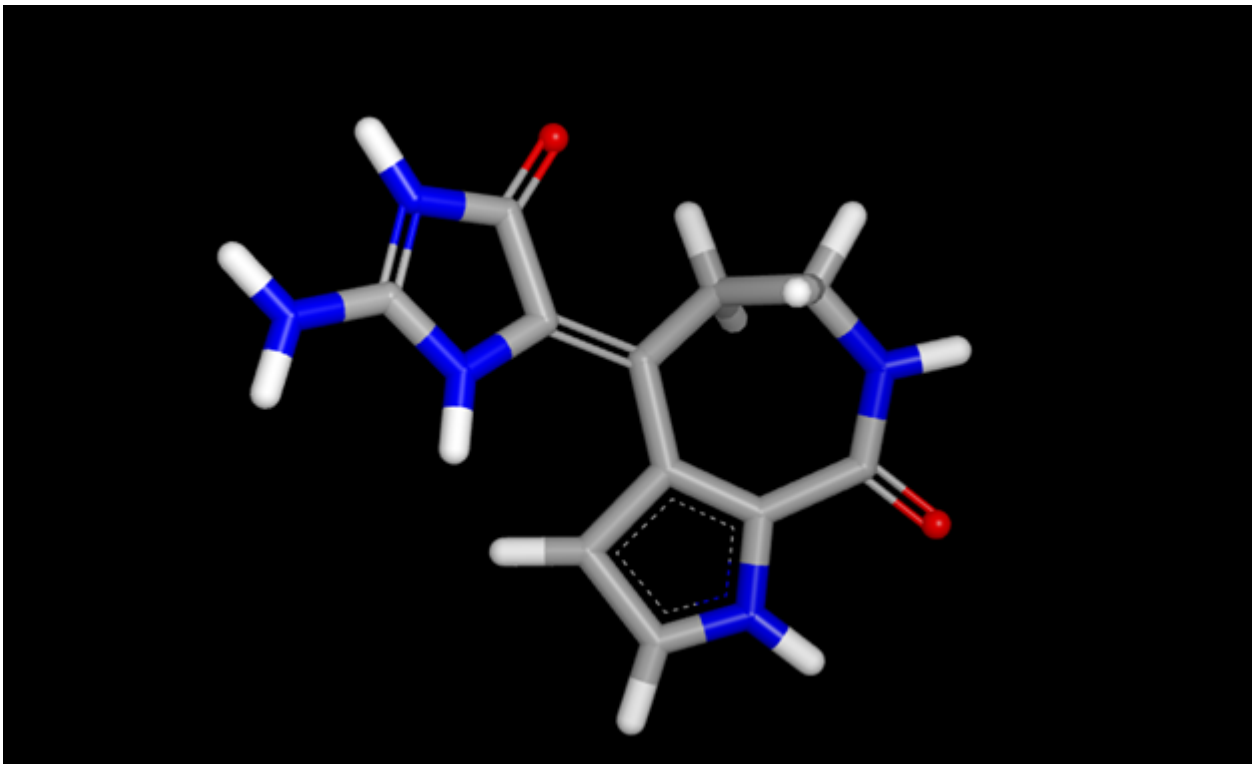
sp3d2

OEHydrogenVisibility

The OEHydrogenVisibility namespace encodes symbolic constants representing the different depiction of hydrogens for the *OE3DMolStyle*. The constants can be used as an argument for the *OE3DMolStyle.SetHydrogenVisibility* function in *OE3DMolStyle*.

This namespace contains constants.

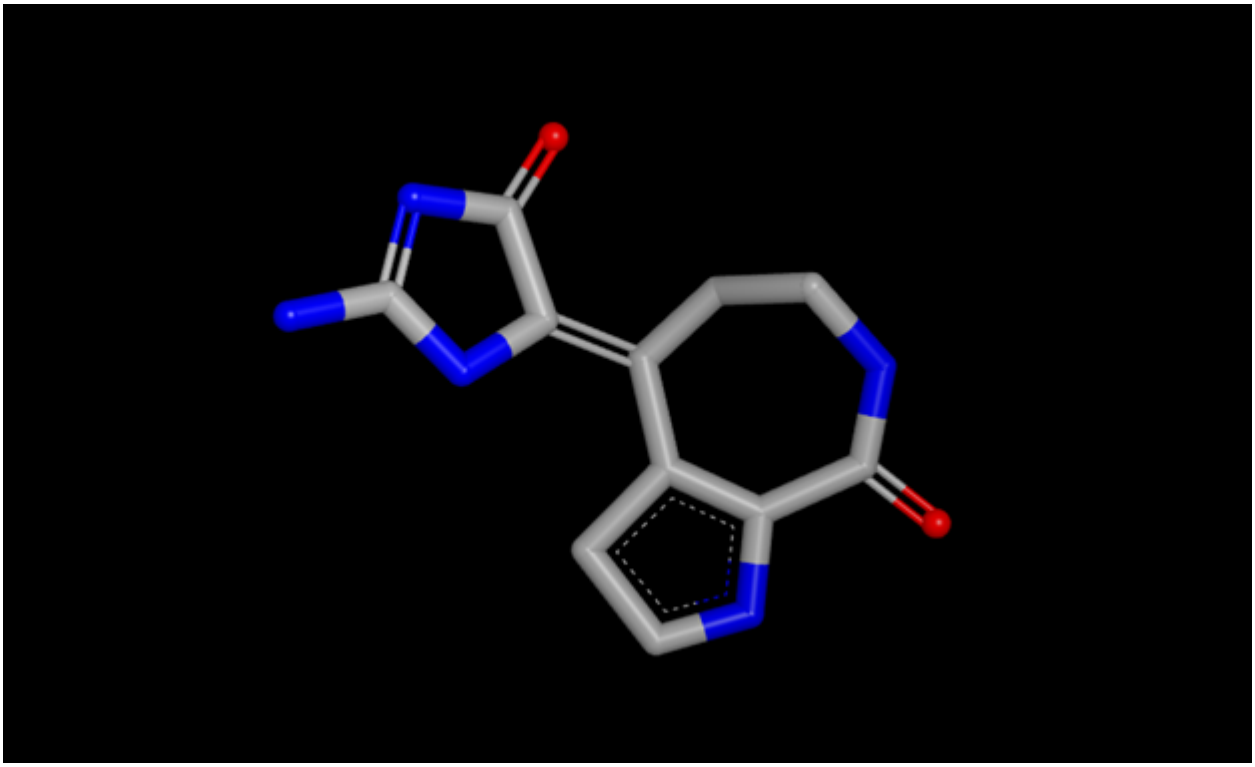
All



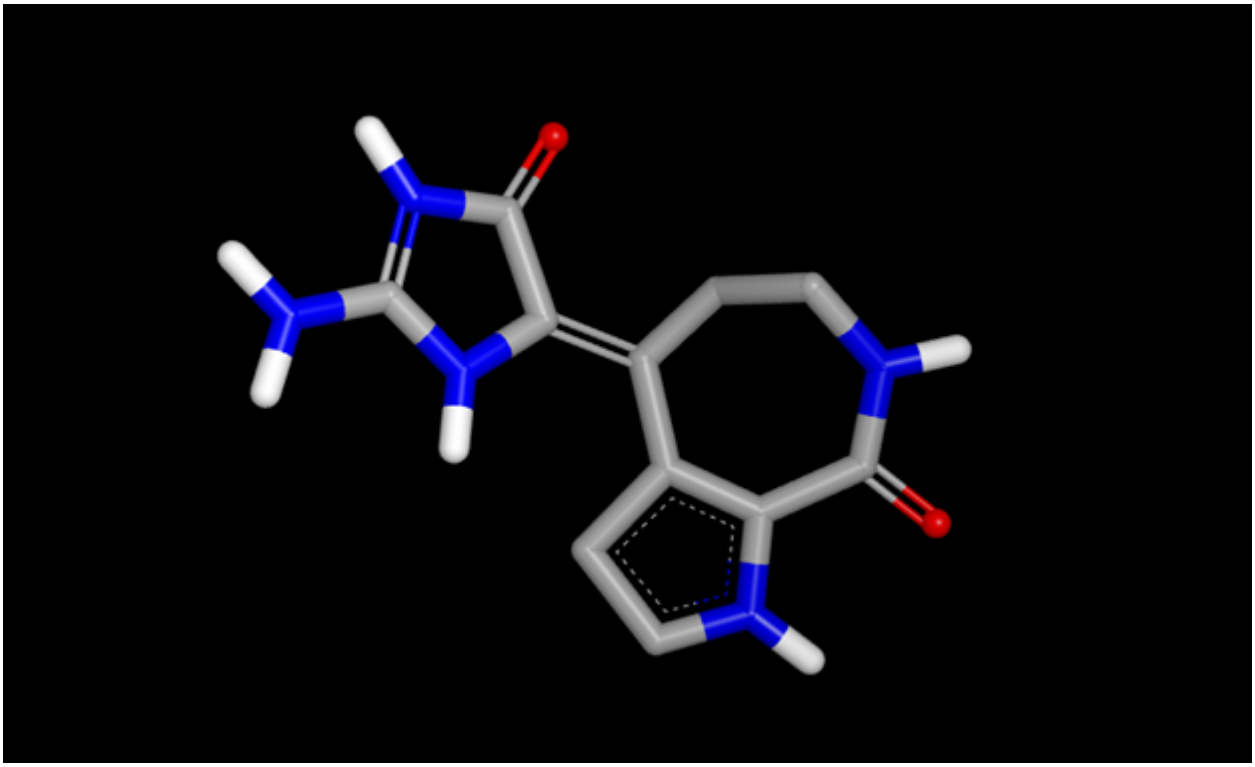
BEGIN

END

Off



Polar



OEIFlavor

The *OEIFlavor* namespace encodes symbolic constants used as bit-masks to indicate how to write various file formats by **OEChem TK**.

> **Note**
>
> A very important introduction (with examples) as to how to manipulate these bit-masks is given *Flavored Input and Output*

The *OEIFlavor.Generic* namespace within *OEIFlavor* defines control bit-masks that are common to all of the input formats. Although these generic bits are common to all file formats, they may be specified independently for each file format.

> **See also**
>
> - *oemolistream.SetFormat* method to set the flavor of an input molecule stream
> - *oemolstreambase.GetFormat* method to retrieve the flavor of a molecule stream

The interpretation of many of these flavor values, is identical to those passed to the corresponding **OEChem TK** low-level file format readers.

This namespace contains the following namespaces:

OEIFlavor::CAN

The *OEIFlavor.CAN* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.CAN* format file by the *OEParseSmiles* function.

This namespace contains the following constants:

Default

No special options are set.

OEIFlavor::CDX

The *OEIFlavor.CDX* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.CDX* format file by the *OEReadCDXFile* low-level file reader.

This namespace contains the following constants:

SuperAtom

Controls whether nested CDX fragment objects, which represent superatom abbreviation groups, are expanded and read with their atoms parented to the enclosing atom. When this flag is set (the default), abbreviations such as `Ph` or `OMe` embedded in a CDX structure are interpreted as substructures.

Default

Same as the *OEIFlavor.CDX.SuperAtom* constant.

OEIFlavor::CIF

The *OEIFlavor.CIF* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.CIF* format file by the *OEReadCIFFile* low-level file reader.

This namespace contains the following constants, that mimic the XYZ reader.

FormalCrg

This flag indicates that high-level molecular file reader calls the *OEAssignFormalCharges* function to assign formal charges.

ImplicitH

This flag indicates that high-level molecular file reader calls the *OEAssignImplicitHydrogens* function to assign implicit hydrogens.

BondOrder

This flag indicates that high-level molecular file reader calls the *OEPerceiveBondOrders* function to perceive bond orders.

Rings

The reader stores excluded sites and sites left after filtering. The filtering of these sites is based on the flavors listed below.

OccFilterOneHalf

If occupancy is noted in the CIF file, only the sites with occupancy larger than one half is kept. If occupancy is not defined in the CIF file, the occupancies are set to 1 to prevent filtering.

RemovePBCImages

Each site read is checked to see if it is equal to a previously read site up to periodic wrapping in fractional coordinates.

NormalizeHydPos

Hydrogen positions and bond lengths are normalized if necessary.

BondHydToClosest

Enforces that hydrogen atoms are bonded to their closest heavy atom in space.

RemoveQuestionMarkInLabel

Sites with a question mark after atom labels are assigned are filtered out. These extra sites are sometimes alternative positions of existing sites.

Default

Combination of: *OEIFlavor.CIF.FormalCrg*, *OEIFlavor.CIF.ImplicitH*, *OEIFlavor.CIF.BondOrder*, *OEIFlavor.CIF.Rings*, *OEIFlavor.CIF.OccFilterOneHalf*, *OEIFlavor.CIF.RemovePBCImages*, *OEIFlavor.CIF.NormalizeHydPos*, *OEIFlavor.CIF.BondHydToClosest*, *OEIFlavor.CIF.RemoveQuestionMarkInLabel* flags.

OEIFlavor::CSV

This namespace contains constants for modifying the behavior of how the *OEFormat.CSV* file format is read.

See also

CSV File Format

Header

Attempt to interpret the first line of the `.csv` file as a header line containing column names. The header line will be parsed into an *OECSVHeader* object and stored on the *oemolistream* object, retrievable by *oemolstreambase.GetCSVHeader*.

If this flag is not set, or the first column of the first line does not contain a valid SMILES string, the *OECSVHeader* will be initialized with the number of columns present in the first line of the file. The columns will then be named and parsed in the following order: SMILES, TITLE, OE_CSV_COLUMN_1, OE_CSV_COLUMN_2, ..., OE_CSV_COLUMN_N.

Default

Set to include *OEIFlavor.CSV.Header* to automatically parse column names into molecule SD data.

DEFAULT

Synonym for *OEIFlavor.CSV.Default*.

OEIFlavor::CXSMILES

The *OEIFlavor.CXSMILES* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEParseSmiles* function.

This namespace contains the following constants:

Strict

Enables strict parsing mode for CXSMILES strings. The parser applies more rigorous sanity checking, rejecting malformed SMILES strings that would be accepted by the non-strict parser. For example, C==C is accepted by default but rejected in strict mode.

Canon

When set, the parser assumes the SMILES string is well-formed and omits the usual kekulization step performed by *OEKekulize*. This can speed up parsing of large databases, but bond orders may not be correctly assigned for aromatic molecules.

DEFAULT

No special parsing options are set. Both *OEIFlavor.CXSMILES.Strict* and *OEIFlavor.CXSMILES.Canon* flags are disabled.

Default

Same as the *OEIFlavor.CXSMILES.DEFAULT* constant.

OEIFlavor::FASTA

The *OEIFlavor.FASTA* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.FASTA* format file by the *OEReadFASTAFile* low-level file reader.

This namespace contains the following constants:

Default

No special options are selected.

CustomResidues

Instructs the FASTA reader to interpret values surrounded in square brackets, `[foo]`, as custom residues. The structure of these custom residues must first be registered with the *OEAddCustomFASTAResidue* function.

> **Warning**
>
> This is an experimental API that may change in the future.

EmbeddedSMILES

Instructs the FASTA reader to interpret values surrounded in square brackets, `[CCC]`, as SMILES strings if a custom residue with the same name can not be found. The SMILES can optionally contain attachment points back to the peptide backbone. For example, `[[R1]CSC[R15]]` specifies to attach to the first and fifteenth residue in the peptide, effectively cyclizing the peptide.

> **Warning**
>
> This is an experimental API that may change in the future.

AllMask

Combination of *OEIFlavor.FASTA.CustomResidues* and *OEIFlavor.FASTA.EmbeddedSMILES* flags.

OEIFlavor::Generic

This namespace contains the following constants:

OEAroModelDaylight

This flag indicates that the Daylight aromaticity model is used to perceive aromaticity by the *OEReadMolecule* high-level reader.

OEAroModelOpenEye

This flag indicates that the OpenEye aromaticity model is used to perceive aromaticity by the *OEReadMolecule* high-level reader.

OEAroModelTripos

This flag indicates that the Tripos aromaticity model is used to perceive aromaticity by the *OEReadMolecule* high-level reader.

OEAroModelMMFF

This flag indicates that the MMFF aromaticity model is used to perceive aromaticity by the *OEReadMolecule* high-level reader.

OEAroModelMDL

This flag indicates that the MDL aromaticity model is used to perceive aromaticity by the *OEReadMolecule* high-level reader.

AroMask

Combination of *OEIFlavor.Generic.OEAroModelDaylight*, *OEIFlavor.Generic.OEAroModelOpenEye*, *OEIFlavor.Generic.OEAroModelTripos*, *OEIFlavor.Generic.OEAroModelMMFF* and *OEIFlavor.Generic.OEAroModelMDL* flags.

See also

- *Aromaticity Models in OEChem TK* section
- *OEAssignAromaticFlags* function

Rings

This flag indicates the rings are perceived by the *OEReadMolecule* high-level reader by calling the *OEFindRingAtomsAndBonds* function.

GenericMask

Combination of *OEIFlavor.Generic.AroMask* and *OEIFlavor.Generic.Rings* flags.

SpecificMask

Bit mask covering all format-specific flavor bits (bits 0-13). This is the complement of *OEIFlavor.Generic.GenericMask*.

DEFAULT

Same as the *OEIFlavor.Generic.Rings* constant. Ring perception is performed after every molecule read by the high-level *OEReadMolecule* reader.

Default

Same as the *OEIFlavor.Generic.DEFAULT* constant.

OEIFlavor::ISM

The *OEIFlavor.ISM* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.ISM* format file by the *OEParseSmiles* function.

This namespace contains the following constants:

Default

No special options are set.

OEIFlavor::MDL

The *OEIFlavor.MDL* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.MDL* format file by the *OEReadMDLFile* low-level file reader.

This namespace contains the following constants:

Default

No special options are set. The *OEIFlavor.MDL.FixBondMarks*, *OEIFlavor.MDL.SuppressEmptyMolSkip*, *OEIFlavor.MDL.SuppressImp2ExpENHSTE*, and *OEIFlavor.MDL.ForceImp2ExpENHSTE* flags are all disabled.

FixBondMarks

This input flavor requests that the reader make an attempt to repair stereocenters that throw warnings due to issues identified by *OEMDLStereoFromBondStereo*. With this input flavor enabled, the warnings will continue to be issued to allow user inspection and verification of the modified bond marks. However, once the corrected structure(s) have been rewritten with any format that captures bond stereomark information, a re-read of the structure(s) should emit fewer stereocenter correction warnings. This input flavor is not enabled for the default *OEFormat.MDL* setting.

SuppressEmptyMolSkip

This input flavor suppresses the default action of skipping empty molecules in the input stream. This may be important in order to recover SDData stored on empty molecule records.

SuppressImp2ExpENHSTE

This input flavor suppresses any enhanced stereogroup implicit to explicit conversions for an input V3000 format molfile.

ForceImp2ExpENHSTE

This input flavor performs implicit to explicit enhanced stereogroup conversions for an input V2000 or V3000 format molfile.

OEIFlavor::MMCIF

The *OEIFlavor.MMCIF* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.MMCIF* format file by the *OEReadCIFFile* low-level file reader.

This namespace contains the following constants.

FormalCrg

This flag indicates that high-level molecular file reader calls the *OEAssignFormalCharges* function to assign formal charges.

ImplicitH

This flag indicates that high-level molecular file reader calls the *OEAssignImplicitHydrogens* function to assign implicit hydrogens.

BondOrder

This flag indicates that high-level molecular file reader calls the *OEPerceiveBondOrders* function to perceive bond orders.

Rings

This flag indicates that high-level molecular file reader calls the *OEFindRingAtomsAndBonds* function to determine ring structure.

Connect

This flag indicates that high-level molecular file reader calls the *OEDetermineConnectivity* function to perceive bond connectivity from atomic coordinates.

SecStruct

This flag indicates that high-level molecular file reader calls the *OEPerceiveSecondaryStructure* function to assign secondary structure.

ALTLOC

This flavor is required when *OEAltLocationFactory* is to be used so that all the atoms with alternate location codes are retained.

DEFAULT

Combination of *OEIFlavor.MMCIF.Connect*, *OEIFlavor.MMCIF.Rings*, *OEIFlavor.MMCIF.BondOrder*, *OEIFlavor.MMCIF.ImplicitH*, *OEIFlavor.MMCIF.SecStruct*, *OEIFlavor.MMCIF.FormalCrg*, flags.

Default

Same as the *OEIFlavor.MMCIF.DEFAULT* constant.

SpruceDefault

Combination of *OEIFlavor.MMCIF.DEFAULT*, *OEIFlavor.MMCIF.ALTLOC*, flags.

AllMask

Combination of *OEIFlavor.MMCIF.DEFAULT* and *OEIFlavor.MMCIF.ALTLOC* flags.

OEIFlavor::MMOD

The *OEIFlavor.MMOD* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.MMOD* format file by the *OEReadMacroModelFile* low-level file reader.

This namespace contains the following constants:

FormalCrg

This flag indicates that high-level molecular file reader calls the *OEAssignFormalCharges* function to assign formal charges.

Default

No special options are set. Formal charge assignment via *OEAssignFormalCharges* is disabled.

OEIFlavor::MOL2

The *OEIFlavor.MOL2* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecular file reader *OEReadMolecule* after reading *OEFormat.MOL2* format file by the *OEReadMol2File* low-level file reader.

This namespace contains the following constants:

M2H

Instructs the MOL2 file reader to treat the file as containing explicit hydrogen atoms. When set, the reader applies additional formal charge and resonance form corrections appropriate for structures with explicit hydrogens. This corresponds to the behavior of the *OEFormat.MOL2H* format.

Forcefield

This specifies a commonly used variant of MOL2, wherein the chemical element for an atom is inferred from the atom name, but not using the second character if it is capitalized. For example:

- CA specifies carbon (not calcium)
- Ca specifies calcium
- NA specifies nitrogen (not sodium)
- Na specifies sodium
- HA specifies hydrogen (not hahnium)

Exceptions were made for atom names beginning with FE, ZN, MG, MN, CL, and BR: these specify chemical elements Fe, Zn, Mg, Mn, Cl, and Br, respectively.

AllMask

Combination of *OEIFlavor.MOL2.M2H* and *OEIFlavor.MOL2.Forcefield* flags.

DEFAULT

No special options are set. Neither *OEIFlavor.MOL2.M2H* nor *OEIFlavor.MOL2.Forcefield* is enabled.

Default

Same as the *OEIFlavor.MOL2.DEFAULT* constant.

OEIFlavor::MOL2H

This namespace contains the following constants:

M2H

Indicates that explicit hydrogen atoms are present in the file. When active, the reader applies formal charge and resonance form corrections appropriate for structures with explicit hydrogens.

AllMask

Same as the *OEIFlavor.MOL2H.M2H* constant; the only flag in this namespace.

DEFAULT

Same as the *OEIFlavor.MOL2H.M2H* constant. Explicit hydrogen treatment is always the default for *OEFormat.MOL2H* format.

Default

Same as the *OEIFlavor.MOL2H.DEFAULT* constant.

OEIFlavor::OEB

This namespace contains the following constants:

Default

No special options are set.

OEIFlavor::OEZ

This namespace contains the following constants:

Default

No special options are set.

OEIFlavor::PDB

This namespace contains the following constants:

TER

Controls whether the PDB file reader should separate consecutive molecules at TER records. *OEIFlavor.PDB.TER* is off by default.

END

Controls whether the PDB file reader should separate consecutive molecules at END records. *OEIFlavor.PDB.END* is on by default.

ENDM

Controls whether the PDB file reader should separate consecutive molecules at ENDM records.

OEIFlavor.PDB.ENDM is on by default.

TerMask

Combination of *OEIFlavor.PDB.TER* and *OEIFlavor.PDB.END* and *OEIFlavor.PDB.ENDM* flags.

Note

By default, only *OEIFlavor.PDB.END* and *OEIFlavor.PDB.ENDM* are on (and *OEIFlavor.PDB.TER* is off), meaning that different chains are read into the a single molecule, but different NMR models and concatenated PDB files are treated as sequential molecules.

If *OEIFlavor.PDB.END*, *OEIFlavor.PDB.ENDM* and *OEIFlavor.PDB.TER* are all turned off, all of the ATOM and HETATM records in a single PDB file will be read into a single *OEMolBase*.

ALL

It used to indicate that all of the atom records in the input file should be read into the *OEMolBase*. By default, the *OEReadPDBFile* function ignores/omits ATOM and HETATM records that represent pseudo or dummy atoms and/or alternate conformations. Without *OEIFlavor.PDB.ALL*, the PDB file reader ignores all atoms whose alternate location indicator is other than `' '`, `A` or `1`, all atoms with atom names beginning `" Q"`, all atoms with residue name `"DUM"`, and all atoms with coordinates 9999.000, 9999.000, 9999.000 (as used by XPLOR/CNS to represent dummy atoms).

ALTLOC

Combination of *OEIFlavor.Generic.Default*, *OEIFlavor.PDB.Default*, and *OEIFlavor.PDB.ALL* flags. This flavor is required when *OEAltLocationFactory* is to be used so that all the atoms with alternate location codes are retained.

DATA

Controls whether the PDB file reader preserves the PDB file's header data as OEMolBase's generic data. This option currently preserves the following records:

- Title section: AUTHOR, CAVEAT, COMPND, EXPDTA, HEADER, JRNL, KEYWDS, OBSLTE, SOURCE, SPRSDE, REMARK, REVDAT, TITLE
- Primary structure section: DBREF, MODRES, SEQRES, SEQADV
- Heterogen section: FORMUL, HET, HETNAM, HETSYM
- Secondary structure section: HELIX, SHEET, TURN
- Connectivity annotation section: SSBOND
- Miscellaneous features section: SITE
- Crystallographic and coordinate transformation section: CRYST1, MTRIX1, MTRIX2, MTRIX3, ORIGX1, ORIGX2, ORIGX3, SCALE1, SCALE2, SCALE3

> **Note**
>
> Modifying or reordering the molecule may invalidate the atom serial numbers used in some of the PDB records.

CHARGE

It is used to indicate that the contents of the b-factor column in the input PDB file contains a partial charge, and should be stored in the 'partial charge' property of an atom, instead of the 'b-factor' property. This value can then be retrieved using the *OEAtomBase.GetPartialCharge* method.

RADIUS

It is used to indicate that the contents of the occupancy column in the input PDB file contains a radius, and should be stored in the 'radius' property of an atom instead of the 'occupancy' property. This value can then be retrieved using the *OEAtomBase.GetRadius* method.

DELPHI

Combinations of *OEIFlavor.PDB.CHARGE* and *OEIFlavor.PDB.RADIUS* flags.

FORMALCHARGE

Controls whether the PDB reader should read the formal charge of the atom from column 79 and 80. Note, this flavor is not turned on default because the subsequent perception steps would not take this formal charge into account when filling valences with bonds and hydrogens.

SecStruct

This controls whether protein secondary structure is perceived after a file is read in. It is on by default. If this flavor is off, secondary structure information specified in the input PDB file (through SHEET and HELIX records) is retained.

BasicMask

Combination of *OEIFlavor.PDB.TER*, *OEIFlavor.PDB.END*, *OEIFlavor.PDB.ENDM*, *OEIFlavor.PDB.ALL*, *OEIFlavor.PDB.DATA*, *OEIFlavor.PDB.CHARGE*, *OEIFlavor.PDB.RADIUS* and *OEIFlavor.PDB.FORMALCHARGE* flags. These are the flags honored by the low-level *OEReadPDBFile* reader.

FormalCrg

> **Note**
>
> This flag is only relevant in the high-level file reader (*OEReadMolecule*) and not in *OEReadPDBFile*.

ImplicitH

> **Note**
>
> This flag is only relevant in the high-level file reader (*OEReadMolecule*) and not in *OEReadPDBFile*.

BondOrder

> **Note**
>
> This flag is only relevant in the high-level file reader (*OEReadMolecule*) and not in *OEReadPDBFile*.

Rings

> **Note**
>
> This flag is only relevant in the high-level file reader (*OEReadMolecule*) and not in *OEReadPDBFile*.

Connect

> **Note**
>
> This flag is only relevant in the high-level file reader (*OEReadMolecule*) and not in *OEReadPDBFile*.

ExtraMask

> **Note**
>
> This flag is only relevant in the high-level file reader (*OEReadMolecule*) and not in *OEReadPDBFile*.

AllMask

Combination of *OEIFlavor.PDB.BasicMask* and *OEIFlavor.PDB.ExtraMask* flags.

DEFAULT

Combination of *OEIFlavor.PDB.Connect*, *OEIFlavor.PDB.Rings*, *OEIFlavor.PDB.BondOrder*, *OEIFlavor.PDB.ImplicitH*, *OEIFlavor.PDB.FormalCrg*, *OEIFlavor.PDB.ENDM* and *OEIFlavor.PDB.END* *OEIFlavor.PDB.DATA*, flags.

Default

Same as the *OEIFlavor.PDB.DEFAULT* constant.

SpruceDefault

Combination of *OEIFlavor.PDB.DEFAULT*, *OEIFlavor.PDB.ALTLOC*, flags.

OEIFlavor::RDF

The *OEIFlavor.RDF* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.RDF* format file by the *OEReadMDLFile* low-level file reader.

Default

No special options are set. The *OEIFlavor.RDF.FixBondMarks*, *OEIFlavor.RDF.SuppressEmptyMolSkip*, *OEIFlavor.RDF.SuppressImp2ExpENHSTE*, and *OEIFlavor.RDF.ForceImp2ExpENHSTE* flags are all disabled.

FixBondMarks

This input flavor requests that the reader make an attempt to repair stereocenters that throw warnings due to issues identified by *OEMDLStereoFromBondStereo*. With this input flavor enabled, the warnings will continue to be issued to allow user inspection and verification of the modified bond marks. However, once the corrected structure(s) have been rewritten with any format that captures bond stereomark information, a re-read of the structure(s) should emit fewer stereocenter correction warnings. This input flavor is not enabled for the default *OEFormat.RDF* setting.

SuppressEmptyMolSkip

This input flavor suppresses the default action of skipping empty molecules in the input stream. This may be important in order to recover SDData stored on empty molecule records.

SuppressImp2ExpENHSTE

This input flavor suppresses any enhanced stereogroup implicit to explicit conversions for an input V3000 format RD-File.

ForceImp2ExpENHSTE

This input flavor performs implicit to explicit enhanced stereogroup conversions for an input RDFile.

OEIFlavor::SDF

The *OEIFlavor.SDF* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.SDF* format file by the *OEReadMDLFile* low-level file reader.

This namespace contains the following constants:

Default

No special options are set. The *OEIFlavor.SDF.FixBondMarks*, *OEIFlavor.SDF.SuppressEmptyMolSkip*, *OEIFlavor.SDF.SuppressImp2ExpENHSTE*, and *OEIFlavor.SDF.ForceImp2ExpENHSTE* flags are all disabled.

FixBondMarks

This input flavor requests that the reader make an attempt to repair stereocenters that throw warnings due to issues identified by *OEMDLStereoFromBondStereo*. With this input flavor enabled, the warnings will continue to be issued to allow user inspection and verification of the modified bond marks. However, once the corrected structure(s) have been rewritten with any format that captures bond stereomark information, a re-read of the structure(s) should emit fewer stereocenter correction warnings. This input flavor is not enabled for the default *OEFormat.SDF* setting.

SuppressEmptyMolSkip

This input flavor suppresses the default action of skipping empty molecules in the input stream. This may be important in order to recover SDData stored on empty molecule records.

SuppressImp2ExpENHSTE

This input flavor suppresses any enhanced stereogroup implicit to explicit conversions for an input V3000 format SD-File.

ForceImp2ExpENHSTE

This input flavor performs implicit to explicit enhanced stereogroup conversions for an input V2000 or V3000 format SDFile.

OEIFlavor::SKC

The *OEIFlavor.SKC* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.SKC* format file by the *OEReadSketchFile* low-level file reader.

This namespace contains the following constants:

Default

No special options are set.

OEIFlavor::SMI

The *OEIFlavor.SMI* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEParseSmiles* function.

This namespace contains the following constants:

Strict

Enables strict parsing mode when processing SMILES strings. The parser applies more rigorous sanity checking, rejecting malformed strings that the non-strict parser would accept. For example, `C==C` is accepted by default but rejected in strict mode.

Canon

When set, the parser assumes the SMILES string is well-formed and omits the usual kekulization step performed by *OEKekulize*. This can speed up parsing of large databases, but bond orders may not be correctly assigned for aromatic molecules.

DEFAULT

No special options are set. Both *OEIFlavor.SMI.Strict* and *OEIFlavor.SMI.Canon* flags are disabled.

Default

Same as the *OEIFlavor.SMI.DEFAULT* constant.

OEIFlavor::USM

The *OEIFlavor.USM* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEParseSmiles* function.

This namespace contains constants.

Default

No special options are set.

OEIFlavor::XYZ

The *OEIFlavor.XYZ* namespace encodes symbolic constants used as bit-masks that control the processing performed by the high-level molecule file reader *OEReadMolecule* after reading *OEFormat.XYZ* format file by the *OEReadXYZFile* low-level file reader.

This namespace contains the following constants:

FormalCrg

This flag indicates that high-level molecular file reader calls the *OEAssignFormalCharges* function to assign formal charges.

ImplicitH

This flag indicates that high-level molecular file reader calls the *OEAssignImplicitHydrogens* function to assign implicit hydrogens.

BondOrder

This flag indicates that high-level molecular file reader calls the *OEPerceiveBondOrders* function to perceive bond orders.

Rings

This flag indicates that high-level molecular file reader calls the *OEFindRingAtomsAndBonds* function to determine ring structure.

Connect

This flag indicates that high-level molecular file reader calls the *OEDetermineConnectivity* function to perceive bond connectivity from atomic coordinates. Since *OEFormat.XYZ* files contain only atomic positions without explicit bond information, this flag is required to establish bonding.

ExtraMask

Combination of *OEIFlavor.XYZ.Connect*, *OEIFlavor.XYZ.Rings*, *OEIFlavor.XYZ.BondOrder*, *OEIFlavor.XYZ.ImplicitH* and *OEIFlavor.XYZ.FormalCrg* flags.

Default

Same as the *OEIFlavor.XYZ.ExtraMask* constant.

OEInteractionStyle

The OEInteractionStyle namespace encodes symbolic constants representing the different hydrogen interaction styles for the *OE3DMolStyle*.

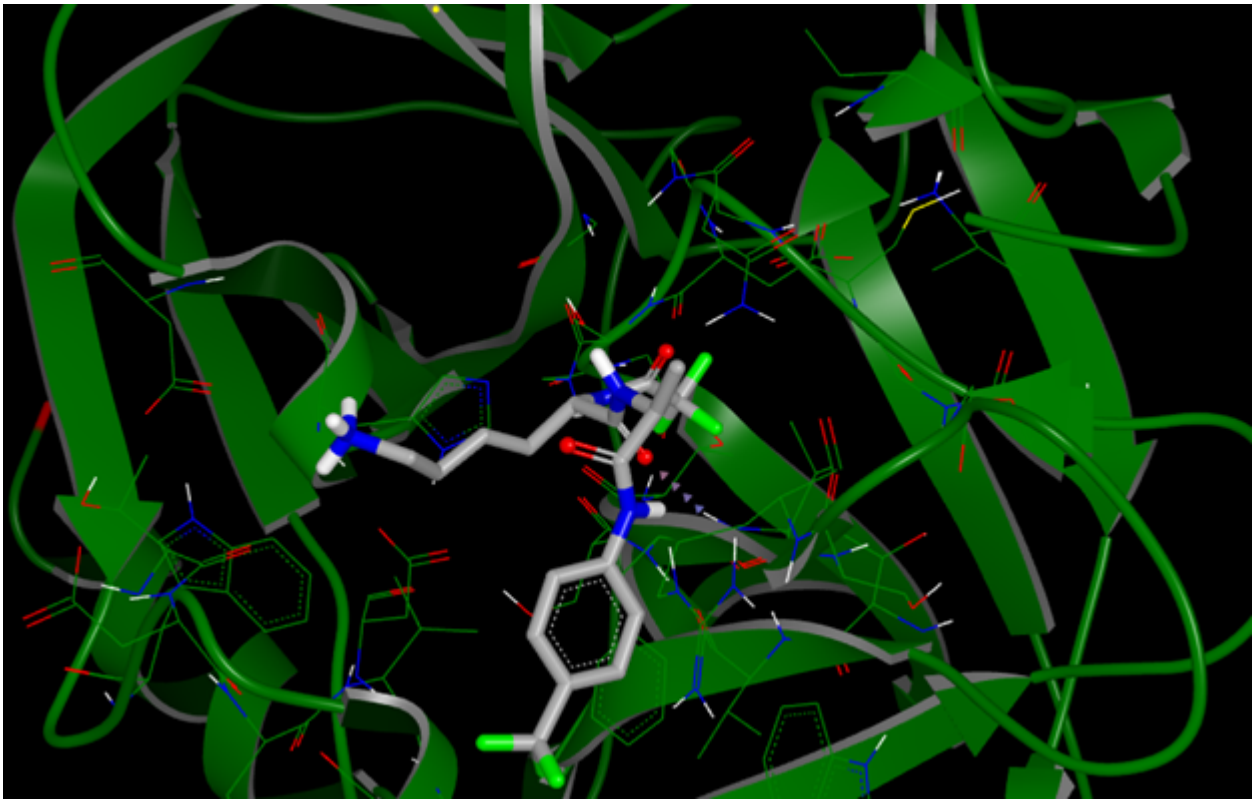
This namespace contains constants.

BEGIN

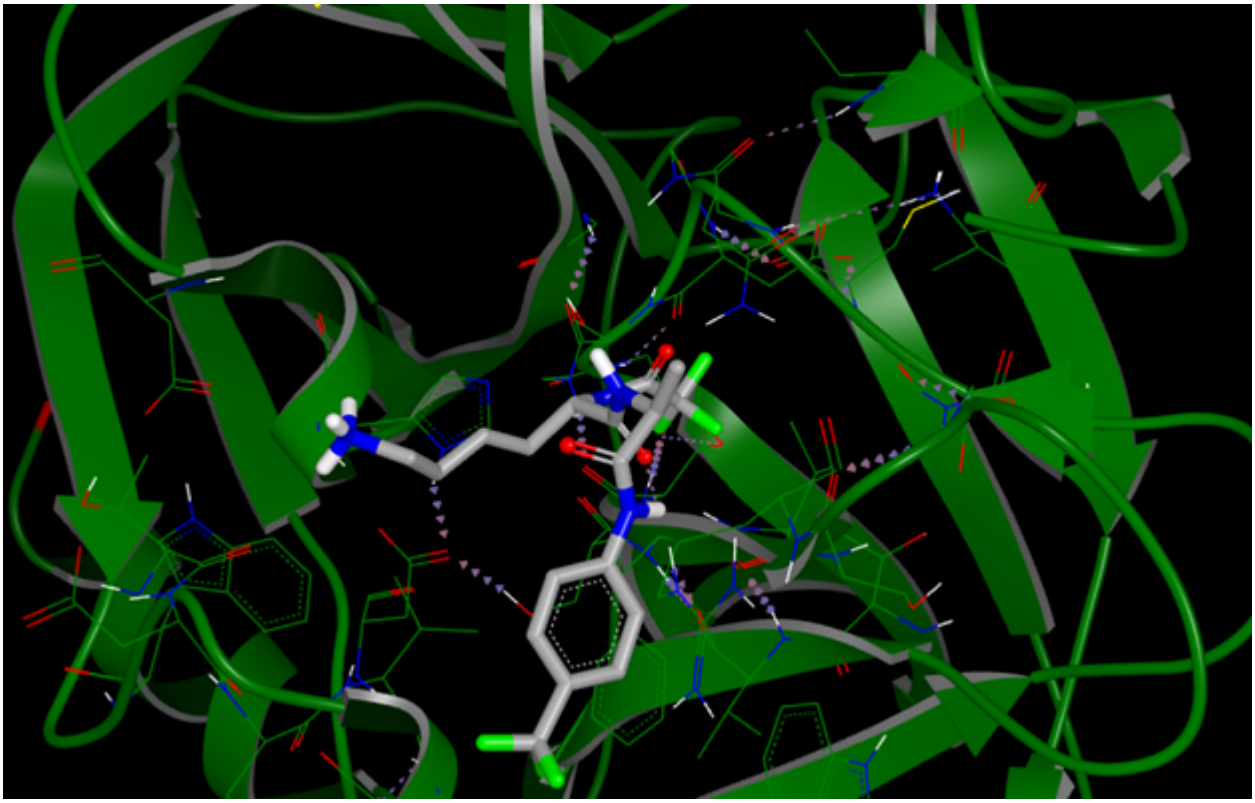
Bumps

END

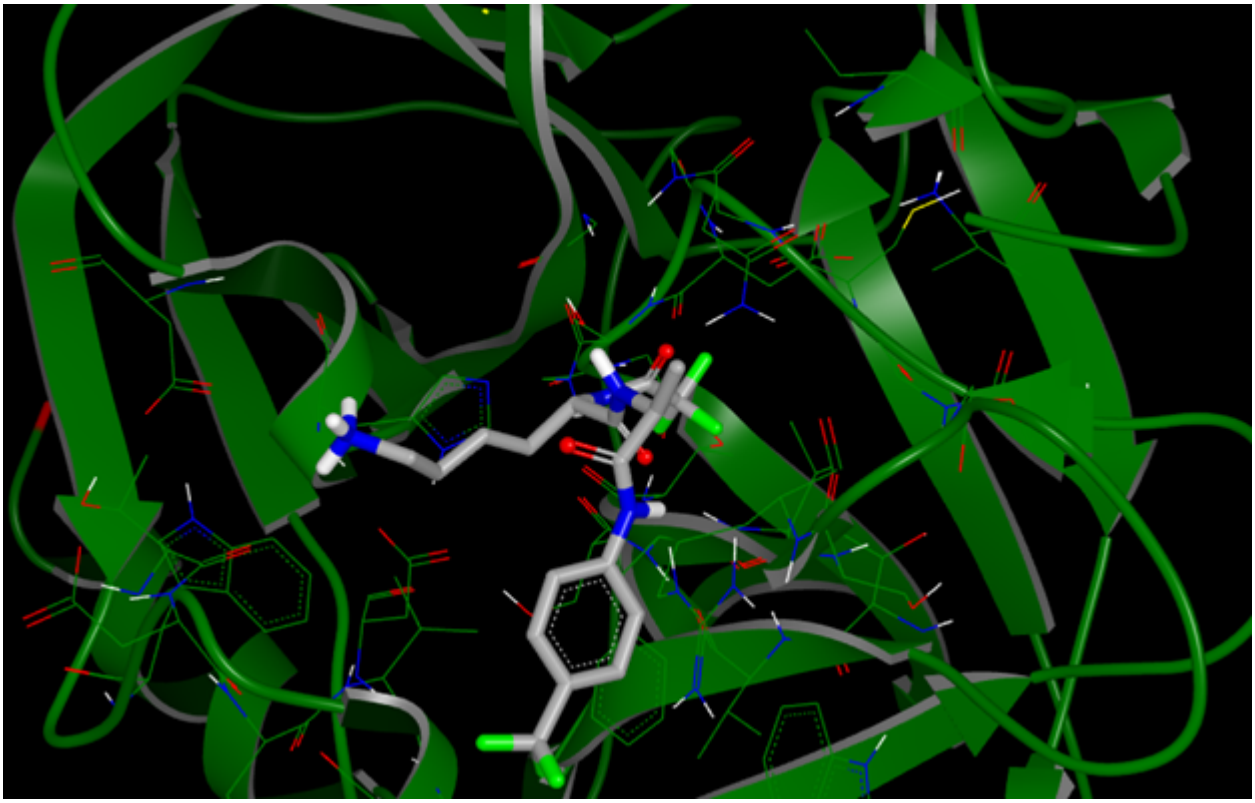
IntermolecularHBonds



IntramolecularHBonds



Off



UnsatisfiedHBonds

OELingoType

This namespace contains constants.

Undefined

IsoSmiMap

Default

OEMCMolType

The *OEMCMolType* namespace encodes symbolic constants representing specific *OEMCMolBase* class implementation types.

See also

- *OEMol.Constructors* function

- *OENewMCMolBase* function

This namespace contains the following constants:

Cartesian

The same as *OEMCMolType.FloatCartesian* and *OEMCMolType.OEDefault*. The default *OEMol* implementation. Conformer coordinates are stored in 32-bit `float` precision representation.

DoubleCartesian

Conformer coordinates will be stored in 64-bit `double` precision. Useful by applications requiring a higher degree of precision, like numerical optimization or quantum mechanics. Writing out molecules of this type to *OEFormat.XYZ* or *OEFormat.OEB* file formats will maintain this level of precision. Even rotor-offset-compressed OEB files can maintain precision to the 15th decimal place when using a `DoubleCartesian` *OEMol*.

FloatCartesian

The same as *OEMCMolType.Cartesian* and *OEMCMolType.OEDefault*. The default *OEMol* implementation. Conformer coordinates are stored in 32-bit `float` precision representation.

HalfFloatCartesian

Conformer coordinates will be stored in 16-bit *OEHalfFloat* precision defined by the IEEE 754-2008 standard. Useful for applications that need to conserve memory or file size. Writing out molecules of this type to *OEFormat.OEB* will only preserve 4 decimal places of precision in conformer coordinates. Rotor-offset-compressed OEB files will only preserve up to the 2nd decimal place for conformer coordinates.

LongDoubleCartesian

Conformer coordinates will be stored in `long double` precision. `long double` precision is implementation defined, only defined to be greater than or equal to 64-bits. Writing out molecules of this type to *OEFormat.OEB* first convert to `double` precision and only 64-bits of precision will be preserved.

OEDBMCMol

An implementation of *OEMCMolBase* designed to have a very small memory footprint when compressed using the *OEMolBase.Compress* method. Compressed molecules are very fragile, the only operations that are valid is destruction, *OEMolBase.Clear*, and *OEMolBase.UnCompress*. Using any other method is undefined. Users should uncompress the molecule before working with it.

OEDefault

The same as *OEMCMolType.Cartesian* and *OEMCMolType.FloatCartesian*. The default *OEMol* implementation. Conformer coordinates are stored in 32-bit `float` precision representation.

Undefined

Used to signal errors and undefined behavior.

MaxType

How many unique numerical constants are in this namespace.

OEMCSType

The *OEMCSType* namespace encodes symbolic constants representing MCS search types.

See also

- *OEMCSSearch* class
- *Exhaustive and approximate MCSS* section

This namespace contains the following constants:

Exhaustive

This constraint indicates that the MCS search is based on a back-tracking algorithm in which the atom mapping of two structures is systematically extended.

Approximate

This constraint indicates that the MCS search is based on traversing through pre-defined paths of the query structure and trying to map the visited atom into the target.

Default

Same as the *OEMCSType.Exhaustive* constant.

OEMDLQueryOpts

The *OEMDLQueryOpts* namespace encodes symbolic constants used as bit-masks to indicate how to interpret MDL query files or MDL reaction imported by *OEReadMDLQueryFile* and *OEReadMDLReactionQueryFile* functions, respectively.

This namespace contains the following constants:

Default

Only constraints explicitly specified in the MDL file are added to the *OEQMolBase* query structure with the the `OEMDLQueryOpts.Optimize` option.

SuppressExplicitH

Controls how the explicit hydrogens of the query are matched to the explicit/implicit hydrogens of the target structures.

See also

- *Supported MDL Query Features* section for the supported MDL query features
- *MDL Query Interpretation* section
- *Explicit Hydrogens* section

AddBondAliphaticConstraint

Controls whether or not map an aliphatic query bond only to aliphatic bonds in the target structure during the substructure search.

See also

- *MDL Query Interpretation* section

AddBondTopologyConstraint

By default, ring bonds in the query structure can only be mapped to ring bonds in the target, and chain bonds in the query can be mapped to both ring and chain bonds.

The `OEMDLQueryOpts.AddBondTopologyConstraint` option enables enforcement of bond topology constraint during substructure search i.e. ring bonds in the query can only be mapped to ring bonds, and chain bonds in the query can only be mapped to chain bonds.

See also

- *MDL Query Interpretation* section

ExactMatchStarAtoms

By default, R-group atoms and * atoms (`OEElemNo.Du`) match like an "any"-atom - matching atoms of any type for both SMARTS and MDL queries. Adding this non-default option flag will force **atoms* that are neither pseudo-atoms nor R-group atoms to match only explicit * atoms in the target.

See also

- *MDL Query Interpretation* section

MatchAtomStereo

By default, atom stereo information is not considered when building query molecules, allowing unconditional match between any atom configuration.

When the `OEMDLQueryOpts.MatchAtomStereo` option is turned on, an S/R atom stereo configuration in the query will match only to any S/R configuration in the target molecule, but not to a R/S (opposite) or an unspecified one. If an atom stereo configuration is undefined in the query, it can match to any atom regardless to its stereo configuration.

See also

- *MDL Query Interpretation* section

MatchIsotope

By default, the isotopic information (*i.e.* lines starting with `M ISO` in the MDL query file) is ignored.

When the `OEMDLQueryOpts.MatchIsotope` option is turned on, a query isotope atoms can be match to a target atom only if it has the same atomic mass. If the query atom has no specified isotope number, it will still match to any target atom regardless of its atomic mass.

See also

- *MDL Query Interpretation* section

Optimize

When the `OEMDLQueryOpts.Optimize` option is turned on, the atom and bond expressions are optimized for substructure search.

ReactionQuery

MDL reaction files are interpreted differently from MDL query files. The usage of `OEMDLQueryOpts.ReactionQuery` option is necessary when a MDL reaction is imported with the `OEReadMDLReactionQueryFile` function.

See also

- *MDL Reaction Query File* section

OEMMFFType

This namespace contains constants.

UNK

CR

CC

CSP2

CO

CN

CGD

COR

CON

COO

COON

COOO

COS

CS

CSN

CSO2

CSO

CSS

CP

CSP

C

DUMMY

HC

HSI

O

OR

OCO

OCC

OCN

OCS

ONO2

ONO

OSO3

OSO2

OSO

OSO_

OS

OPO3

OPO2

OPO

OP

OC

OCN_

OCR

OCO_

ON

OS_

OS__

NR

NC

NN

NCO

NCS

NNC

NNN

F

CL

Cl

BR

Br

I

S

SC

SO

SN

SO2

SO2N

SO3

SO4

SO2_

SNO

SI

CR4R

HOR

HO

CR3R

HNR

HPYL

H3N

HNOX

HOCO

HOP

PO4

PO3

PO2

PO

PTET

P

HNC

HNN

HNCO

HNCS

HNCC

HNCN

HNNC

HNNN

HNSO

HNC_

HSP2

HOCC

HOCN

CR4E

HOH

O2CM

OXN

ON_

O2N

O2NO

O3N

OS___

O2S

O3S

O4S

OSMS

OP_

O2P

O3P

O4P

O4CL

O4CI

HOS

NR_

OM

OM2

HNR_

HIM

HPD_

HNN_

HNC__

HGD_

CB

NPYD

NPYL

NCC

NCN

NCP

NCC_

CO2M

CS2M

NSP

NSO2

NSO3

NCN_

STHI

NO2

NO3

NO

NAZT

NSO

O_

HO_

O__

HO__

N_

NC_

NN_

NCN__

NGD_

CGD_

CNN_

NPD_

OFUR

C__

NR__

NM

C5A

C5B

N5A

N5B

N2OX

N3OX

NPOX

OH2

HS

HSN

HP

SP

SM

SSMO

SO2M

SSOM

SO_

PC

N5M

CLO4

C5

N5

CIM

NIM

N5A_

N5B_

N5_

N5AX

N5BX

N5OX

FE2

FE3

F_

CL_

BR_

LI

NA

K

ZN2

CA2

CU1

CU2

MG2

HOM

HOCS

HON

HX

HNO

HN5_

NPO2

NPO3

HNPO

HNO2

HNM

CONN

S2CM

SE

Se

B

HSE

HSe

CION

MAXTYPE

OEMModType

This namespace contains constants.

C1

C2

C3

CA

CB

CC

CD

CE

CF

CM

CP

CR

C0

O2

O3

OA

OM

OW

OP

OQ

O0

N1

N2

N3

NA

NB

NC

ND

N4

N5

NE

NF

NG

NH

NI

NM

NP

N0

H1

H2

H3

H4

H5

H0

S1

SA

SM

S0

P0

B2

B3

F0

Cl

Br

I0

Si

Du

Z0

Lp

Li

Na

K0

Rb

Cs

Ca

Ba

Mg

M2

M3

M4

M5

M6

M7

f2

f3

o2

o3

n2

n3

c1

c2

Zn

m3

m4

m5

m6

SP

S2

Cm

MAXTYPE

OEMolBaseType

The *OEMolBaseType* namespace encodes symbolic constants representing specific *OEMolBase* class implementation types.

See also

- *OEGraphMol* constructor
- *OENewMolBase* function

This namespace contains the following constants:

Undefined

Used to signal errors.

OEDefault

The default molecule implementation used by OEChem.

OEDBMol

A molecule implementation designed to have a very small memory footprint. *OEMolBase.Compress* can be called to dramatically reduce memory consumption. However, *OEMolBase.UnCompress* must be called to make the molecule usable again.

OEMiniMol

A molecule implementation designed to have a small memory footprint, however, the molecule is still usable for molecular algorithms, i.e., substructure searching. This molecule implementation has been tuned for high performance in-memory substructure searching, while maintaining a small memory footprint. *OEMolBase.Compress* will reduce memory consumption, however, it will put the molecule in a "read-only" state. *OEMolBase.UnCompress* will make the molecule writeable again.

Warning

OEGroupBase information will be lost when compressing molecules that are constructed with the *OEMolBaseType.OEMiniMol* type.

MaxType

The maximum numerical value possible from this namespace.

OENearestNbrsMethod

This namespace contains constants.

This namespace contains constants representing various heuristics used to search for nearest neighbors.

See also

- *OEGetNearestNbrs* function
- *OENearestNbrs* class

Default

The default method is *OENearestNbrsMethod.Auto*.

Auto

When using the *OENearestNbrsMethod.Auto* constant, the algorithm that is utilized to search the nearest neighbors will be determined by the size of the molecule.

AllByAll

This represent a method that is suitable for searching nearest neighbors in small molecules.

LocalHash

This represent a method that is suitable for searching nearest neighbors in large molecules such as proteins.

ZCoordSort

This represent a method that is suitable for searching nearest neighbors in mid-sized molecules.

OEOFlavor

The *OEOFlavor* namespace encodes symbolic constants used as bit-masks to indicate how to read various file formats by **OEChem TK**.

The *OEOFlavor.Generic* namespace within *OEOFlavor* defines control bit-masks that are common to all of the output formats. Although these generic bits are common to all file formats, they may be specified independently for each file format.

See also

- *oemolostream.SetFormat* method to set the flavor of an output molecule stream
- *oemolstreambase.GetFormat* method to retrieve the flavor of a molecule stream

The interpretation of many of these flavor values, is identical to those passed to the corresponding **OEChem TK** low-level file format writers.

This namespace contains the following namespaces:

OEOFlavor::CAN

The *OEOFlavor.CAN* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OECreateSmiString* function when writing a molecule into a *OEFormat.CAN* format file.

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the OpenEye model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

AllBonds

Corresponds to the *OESMILESFlag.AllBonds* constant.

AtomMaps

Corresponds to the *OESMILESFlag.AtomMaps* constant.

AtomStereo

Corresponds to the *OESMILESFlag.AtomStereo* constant.

BondStereo

Corresponds to the *OESMILESFlag.BondStereo* constant.

Canonical

Corresponds to the *OESMILESFlag.Canonical* constant.

DEFAULT

Same as the *OEOFlavor.CAN.Default* constant.

Default

Combination of *OEOFlavor.CAN.AtomMaps*, *OEOFlavor.CAN.Canonical* and *OEOFlavor.CAN.RGroups* flags.

EnhStereo

Corresponds to the *OESMILESFlag.EnhStereo* constant.

ExtBonds

Corresponds to the *OESMILESFlag.ExtBonds* constant.

Hydrogens

Corresponds to the *OESMILESFlag.Hydrogens* constant.

ImpHCount

Corresponds to the *OESMILESFlag.ImpHCount* constant.

Isotopes

Corresponds to the *OESMILESFlag.Isotopes* constant.

Kekule

Corresponds to the *OESMILESFlag.Kekule* constant.

RGroups

Corresponds to the *OESMILESFlag.RGroups* constant.

SmiMask

SuperAtoms

Corresponds to the *OESMILESFlag.SuperAtoms* constant.

OEOFlavor::CDX

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the OpenEye model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

Add2D

This flag indicates that 2D coordinates are generated prior to writing a molecule by the *OEWriteMolecule* high-level writer. If the molecule already has 2D coordinates then these coordinates are left intact. In certain cases the 2D layout may result in incorrect stereochemistry in the resulting connection table. In these cases no coordinates are generated.

UnsetBad2DStereo

This flag is used with the Add2D constant. In certain cases the 2D layout may result in incorrect stereochemistry in the output connection table format. When this flag is set the writer will output the generated coordinates and set the stereochemistry of the offending stereocenters to 'Undefined'.

Default

Same as the *OEOFlavor.CDX.Add2D* constant.

OEOFlavor::CIF

Note

If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the OpenEye model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

Default

OEOFlavor::CSV

This namespace contains constants for modifying the behavior of how the *OEFormat.CSV* file format is written.

See also

CSV File Format

Header

This flag indicates whether the first line in the `.csv` should contain the column names for the file. Column names are determined by the SD data present on the first molecule written to the *oemolostream*, regardless of whether this flag is set. The column names are retrievable through the *oemolstreambase.GetCSVHeader* method on the *OECSVHeader* object.

If this flag is not set, the first line of the output `.csv` file will be the first molecule record.

Default

Set to include *OEOFlavor.CSV.Header* to automatically write the column names as the first line in the file.

DEFAULT

Synonym for *OEOFlavor.CSV.Default*.

OEOFlavor::CXSMILES

The *OEOFlavor.CXSMILES* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OECreateSmiString* function when writing a molecule into a *OEFormat.CXSMILES* format file.

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the OpenEye model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

AllBonds

Corresponds to the *OESMILESFlag.AllBonds* constant.

AtomMaps

Corresponds to the *OESMILESFlag.AtomMaps* constant.

AtomStereo

Corresponds to the *OESMILESFlag.AtomStereo* constant.

BondStereo

Corresponds to the *OESMILESFlag.BondStereo* constant.

Canonical

Corresponds to the *OESMILESFlag.Canonical* constant.

DEFAULT

Same as the *OEOFlavor.CXSMILES.Default* constant.

Default

Combination of *OEOFlavor.CXSMILES.AtomMaps*, *OEOFlavor.CXSMILES.AtomStereo*, *OEOFlavor.CXSMILES.BondStereo*, *OEOFlavor.CXSMILES.Canonical*, *OEOFlavor.CXSMILES.Isotopes*, *OEOFlavor.CXSMILES.RGroups* and *OEOFlavor.CXSMILES.EnhStereo* flags.

EnhStereo

Corresponds to the *OESMILESFlag.EnhStereo* constant.

ExtBonds

Corresponds to the *OESMILESFlag.ExtBonds* constant.

Hydrogens

Corresponds to the *OESMILESFlag.Hydrogens* constant.

ImpHCount

Corresponds to the *OESMILESFlag.ImpHCount* constant.

Isotopes

Corresponds to the *OESMILESFlag.Isotopes* constant.

Kekule

Corresponds to the *OESMILESFlag.Kekule* constant.

RGroups

Corresponds to the *OESMILESFlag.RGroups* constant.

SmiMask

SuperAtoms

Corresponds to the *OESMILESFlag.SuperAtoms* constant.

OEOFlavor::FASTA

This namespace contains the following constants:

Default

OEOFlavor::Generic

This namespace contains the following constants:

OEAroModelDaylight

This flag indicates that the Daylight aromaticity model is used to perceive aromaticity by the *OEWriteMolecule* high-level writer.

OEAroModelOpenEye

This flag indicates that the OpenEye aromaticity model is used to perceive aromaticity by the *OEWriteMolecule* high-level writer.

OEAroModelTripos

This flag indicates that the Tripos aromaticity model is used to perceive aromaticity by the *OEWriteMolecule* high-level writer.

OEAroModelMMFF

This flag indicates that the MMFF aromaticity model is used to perceive aromaticity by the *OEWriteMolecule* high-level writer.

OEAroModelMDL

This flag indicates that the MDL aromaticity model is used to perceive aromaticity by the *OEWriteMolecule* high-level writer.

AroMask

Combination of *OEOFlavor.Generic.OEAroModelDaylight*, *OEOFlavor.Generic.OEAroModelOpenEye*, *OEOFlavor.Generic.OEAroModelTripos*, *OEOFlavor.Generic.OEAroModelMMFF* and *OEOFlavor.Generic.OEAroModelMDL* flags.

See also

- *Aromaticity Models in OEChem TK* section
- *OEAssignAromaticFlags* function

Rings

This flag indicates the rings are (re)perceived by the *OEWriteMolecule* high-level reader by calling the *OEFindRingAtomsAndBonds* function.

GenericMask

Combination of *OEIFlavor.Generic.AroMask* and *OEIFlavor.Generic.Rings* flags.

SpecificMask

DEFAULT

Default

Same as the *OEOFlavor.Generic.DEFAULT* constant.

OEOFlavor::INCHI

The *OEOFlavor.INCHI* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OECreateInChI* function when writing a molecule into the *OEFormat.INCHI* format file. These constants translate into InChI library options documented in the InChI API Reference documentation.

See also

OEInChIOptions

This namespace contains the following constants:

Chiral

Corresponds to the `ChiralFlagOn` InChI option.

Default

Equal to *OEOFlavor.INCHI.Stereo* and *OEOFlavor.INCHI.Title*.

FixedHLayer

Corresponds to the `FixedH` InChI option.

Hydrogens

Corresponds to the inverse of the `DoNotAddH` InChI option.

RacemicStereo

Corresponds to the `SRac` InChI option.

ReconnectedMetals

Corresponds to the `RecMet` InChI option.

RelativeStereo

Corresponds to the `SRel` InChI option.

Stereo

Corresponds to the inverse of the `SNon` InChI option.

Title

If this flag is set, then the molecule's title (returned by *OEMolBase.GetTitle* method) is written to the output file.

OEOFlavor::INCHIKEY

The *OEOFlavor.INCHIKEY* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OECreateInChIKey* function when writing a molecule into a *OEFormat.INCHIKEY* format file. These constants translate into InChI library options documented in the InChI API Reference documentation.

This namespace contains the following constants:

Chiral

Corresponds to the `ChiralFlagOn` InChI option.

Default

Equal to *OEOFlavor.INCHIKEY.Stereo* and *OEOFlavor.INCHIKEY.Title*.

FixedHLayer

Corresponds to the `FixedH` InChI option.

Hydrogens

Corresponds to the inverse of the `DoNotAddH` InChI option.

RacemicStereo

Corresponds to the `SRac` InChI option.

ReconnectedMetals

Corresponds to the `RecMet` InChI option.

RelativeStereo

Corresponds to the `SRel` InChI option.

Stereo

Corresponds to the inverse of the `SNon` InChI option.

Title

If this flag is set, then the molecule's title (returned by *OEMolBase.GetTitle* method) is written to the output file.

OEOFlavor::ISM

The *OEOFlavor.ISM* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OECreateSmiString* function when writing a molecule into a *OEFormat.ISM* format file.

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the OpenEye model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

AllBonds

Corresponds to the *OESMILESFlag.AllBonds* constant.

AtomMaps

Corresponds to the *OESMILESFlag.AtomMaps* constant.

AtomStereo

Corresponds to the *OESMILESFlag.AtomStereo* constant.

BondStereo

Corresponds to the *OESMILESFlag.BondStereo* constant.

Canonical

Corresponds to the *OESMILESFlag.Canonical* constant.

DEFAULT

Same as the *OEOFlavor.ISM.Default* constant.

Default

Combination of *OEOFlavor.ISM.AtomMaps*, *OEOFlavor.ISM.AtomStereo*, *OEOFlavor.ISM.BondStereo*, *OEOFlavor.ISM.Canonical*, *OEOFlavor.ISM.Isotopes* and *OEOFlavor.ISM.RGroups* flags.

EnhStereo

Corresponds to the *OESMILESFlag.EnhStereo* constant.

ExtBonds

Corresponds to the *OESMILESFlag.ExtBonds* constant.

Hydrogens

Corresponds to the *OESMILESFlag.Hydrogens* constant.

ImpHCount

Corresponds to the *OESMILESFlag.ImpHCount* constant.

Isotopes

Corresponds to the *OESMILESFlag.Isotopes* constant.

Kekule

Corresponds to the *OESMILESFlag.Kekule* constant.

RGroups

Corresponds to the *OESMILESFlag.RGroups* constant.

SmiMask

SuperAtoms

Corresponds to the *OESMILESFlag.SuperAtoms* constant.

OEOFlavor::MDL

The *OEOFlavor.MDL* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEWriteMDLFile* and the *OEWriteMolecule* functions when writing a molecule into a *OEFormat.MDL* format file.

This namespace contains the following constants:

Add2D

This flag indicates that if a molecule has no coordinates, then 2D coordinates are generated prior to writing a molecule by the *OEWriteMolecule* high-level writer. If the molecule already has 2D or 3D coordinates then these coordinates are left intact. In certain cases the 2D layout may result in incorrect stereochemistry in the resulting connection table. In these cases no coordinates are generated.

UnsetBad2DStereo

This flag is used with the Add2D constant. In certain cases the 2D layout may result in incorrect stereochemistry in the output connection table format. When this flag is set the writer will output the generated coordinates and set the stereochemistry of the offending stereocenters to 'Undefined'.

MCHG

The *OEOFlavor.MDL.MCHG* flag is used to instruct the MDL mol file writer to write `M CHG` and `M RAD` lines for charged/radical atoms in the output connection table, even if the charge and radical values of every atom are within the ranges representable by the MDL atom block. By default, `M CHG` and `M RAD` lines are only used for connection tables that 'overflow' the allowed atom block limits.

MISO

The *OEOFlavor.MDL.MISO* flag is used to instruct the **OEChem TK** MDL mol file writer to write `M ISO` lines for isotopes in the output connection table, even if the isotopic masses of every atom are within the range representable by the MDL atom block. By default, `M ISO` lines are only used for connection tables that 'overflow' the allowed atom block limits.

MRGP

The *OEOFlavor.MDL.MRGP* flag is used to instruct the **OEChem TK** MDL mol file writer to write out `M RGP` entries for each R-group atom, *i.e.* atom with element zero and a non-zero atom map index. These atoms are always written with atomic symbol `R#`.

MV30

Force every molecule to be written in V3000 format. By default, only molecules with more than 999 atoms or enhanced stereo groups will be written in V3000 format.

MDLParity

If this flag is set and *OEMDLHasParity* returns `false` ,then *OEMDLPerceiveParity* function is called to perceive the MDL atom stereo parity fields.

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWriteMDLFile*.

NoParity

If this flag is set, then *OEMDLClearParity* function is called to set all MDL atom stereo parity fields to zero.

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWriteMDLFile*.

CurrentParity

If this flag is set, then the MDL atom stereo parity fields are left intact.

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWriteMDLFile*.

MMask

PMask

Combination of *OEOFlavor.MDL.MDLParity*, *OEOFlavor.MDL.NoParity* and *OEOFlavor.MDL.CurrentParity* flags. The flag controls the processing of the MDL stereo parity bit field of each atom.

SuppressImp2ExpENHSTE

This flavor suppresses any implicit to explicit enhanced stereogroup conversions if the output is V3000 format.

SuppressTimestamps

This flavor suppresses the output of timestamps in MDL format header line(s). This can be useful for generation of simple diffable standards for use in tests.

DEFAULT

Combination of *OEOFlavor.MDL.Add2D*, *OEOFlavor.MDL.MDLParity*, *OEOFlavor.MDL.MRGP*, *OEOFlavor.MDL.MISO* and *OEOFlavor.MDL.MCHG* flags.

Default

Same as the *OEOFlavor.MDL.DEFAULT* constant.

OEOFlavor::MF

The *OEOFlavor.MF* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEWriteMolecule* functions when writing a molecule into a *OEFormat.MF* format file.

This namespace contains the following constants:

Title

If this flag is set, then the molecule's title (returned by *OEMolBase.GetTitle* method) is written to the output file.

DEFAULT

Same as the *OEOFlavor.MF.Title* constant.

Default

Same as the *OEOFlavor.MF.DEFAULT* constant.

OEOFlavor::MMCIF

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the OpenEye model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

PerceiveMetadata

Default

OEOFlavor::MMOD

This namespace contains following constants:

AtomTypes

DEFAULT

Default

Same as the *OEOFlavor.MMOD.DEFAULT* constant.

OEOFlavor::MOL2

> **Note**
>
> Except for MOL2 Flavor Forcefield, if no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the Tripos model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constant:

AtomNames

AtomTypeNames

BondTypeNames

ChargePrecision

This flavor allows for increased precision of partial charges by writing them out to 5 decimal places.

Forcefield

Combination of *OEOFlavor.MOL2.GeneralFFFormat*, *OEOFlavor.MOL2.ChargePrecision*, and *OEOFlavor.MOL2.BondTypeNames* flags

This variant of MOL2 reflects a common use of MOL2 format to embody information for a general forcefield. It combines the *OEOFlavor.MOL2.GeneralFFFormat* so that atom names, atom types, and atom ordering are written out "as is" based on the input molecule, and *OEOFlavor.MOL2.ChargePrecision*, which writes out atomic partial charges to 5 decimal places. Mol2 *OEOFlavor.MOL2.BondTypeNames* are retained from the Tripos standard at this point because they are generally reasonable and historically they don't conflict with the other needs of general forcefield format.

GeneralFFFormat

This variant of MOL2 is commonly used for carrying information for force fields more general than the Tripos forcefield (the MOL2 standard). Here the atom names, atom types, and atom ordering are retained from the input molecule, rather than being changed to comply with the Tripos MOL2 standard. If an atom does not have a user-defined atom name and atom type for all atoms, warnings will be given and no molecule will be written.

Hydrogens

OrderAtoms

Substructure

NameMask

AllMask

DEFAULT

Default

Same as the *OEOFlavor.MOL2.DEFAULT* constant.

OEOFlavor::MOL2H

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the Tripos model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

AtomTypeNames

BondTypeNames

AtomNames

OrderAtoms

Hydrogens

Substructure

NameMask

AllMask

DEFAULT

Default

Same as the *OEOFlavor.MOL2H.DEFAULT* constant.

OEOFlavor::MOPAC

This namespace contains the following constants:

XYZ

CHARGES

AllMask

DEFAULT

Default

Same as the *OEOFlavor.MOPAC.DEFAULT* constant.

OEOFlavor::OEB

This namespace contains the following constants:

Default

OEOFlavor::OEZ

This namespace contains the following constants:

Default

OEOFlavor::PDB

The *OEOFlavor.PDB* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEWritePDBFile* and the *OEWriteMolecule* functions when writing a molecule into a *OEFormat.PDB* format file.

This namespace contains the following constants:

BONDS

Controls whether the OEChem PDB file writer includes CONECT records to describe the connectivity of the molecule. If this flag is given without *OEOFlavor.PDB.ORDERS* all bonds are written as 'single' bonds. *OEOFlavor.PDB.BONDS* is off by default.

ORDERS

Controls whether the PDB file writer should use repeated CONECT records to represent multiple bond orders (*i.e.* double and triple bonds). If this flag is given without *OEOFlavor.PDB.BONDS* only double and triple bonds are written to the file. *OEOFlavor.PDB.ORDERS* is off by default.

BOTH

Including this bit-mask writes bi-directional CONECT records. Conventionally, PDB files list connect records both from source atom to destination atom, and again from destination to source. This duplication doubles the number/size of CONECT records with redundant information, as most readers (if they honor CONECT records at all) can work with uni-directional CONECT records. By default, this flavor is enabled and bi-directional CONECT records are written.

CHARGE

Controls whether the PDB file writer takes the value for the PDB file's B-factor field from the atom's partial charge property. A number of applications, including Delphi, (ab)use the PDB file format specification to store atomic partial charges in the b-factor field rather instead of crystallographically determined anisotropic temperature factors. *OEOFlavor.PDB.CHARGE* is off by default.

RADIUS

Controls whether the PDB file writer takes the value for the PDB file's occupancy field from the atom's radius property. A number of applications, including Delphi, (ab)use the PDB file format specification to store radii in the occupancy field rather instead of crystallographically determined anisotropic occupancies. *OEOFlavor.PDB.RADIUS* is off by default.

TER

Controls whether the PDB file writer terminates each connection table with a TER record rather than an END record. Some dubious molecular graphics software, such as Grasp, requires that multiple compounds be separated by the chain/fragment separator TER, rather than the molecule separator END. *OEOFlavor.PDB.TER* is off by default.

DELPHI

Combination of *OEOFlavor.PDB.CHARGE* and *OEOFlavor.PDB.RADIUS* flags.

ELEMENT

Controls whether the PDB writer writes the chemical symbol right-justified in columns 77-78, as specified by version 2.2 of the PDB format specification. *OEOFlavor.PDB.ELEMENT* is on by default.

FORMALCHARGE

Controls whether the PDB file writer writes the formal charge right-justified (*e.g.* `2+`, `2-`) in columns 79-80, as specified by version 2.2 of the PDB format specification. This flag is off by default. Enabling *OEOFlavor.PDB.FORMALCHARGE* also enables *OEOFlavor.PDB.ELEMENT*.

HETBONDS

Controls whether the PDB writer writes any bond to or from a 'hetero atom' (i.e. atom that has the record type `HETATM` in a PDB file). *OEOFlavor.PDB.HETBONDS* is on by default.

See also

- *OEPreserveResInfo* namespace
- *OEResidue.IsHetAtom* method to retrieve the 'hetero atom' property of an atom in a residue
- *SetHetAtom* method

BasicMask

OEResidues

Note

This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWritePDBFile*.

NoResidues

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWritePDBFile*.

CurrentResidues

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWritePDBFile*.

OrderAtoms

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWritePDBFile*.

AllMask

DEFAULT

Combination of *OEOFlavor.PDB.OrderAtoms*, *OEOFlavor.PDB.OEResidues*, *OEOFlavor.PDB.HETBONDS*, *OEOFlavor.PDB.ELEMENT* and *OEOFlavor.PDB.BOTH* flags.

Default

Same as the *OEOFlavor.PDB.DEFAULT* constant.

OEOFlavor::SDF

The *OEOFlavor.SDF* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEWriteMDLFile* and the *OEWriteMolecule* functions when writing a molecule into a *OEFormat.SDF* format file.

This namespace contains the following constants:

Add2D

This flag indicates that if a molecule has no coordinates, then 2D coordinates are generated prior to writing a molecule by the *OEWriteMolecule* high-level writer. If the molecule already has 2D or 3D coordinates then these coordinates are left intact. In certain cases the 2D layout may result in incorrect stereochemistry in the resulting connection table. In these cases no coordinates are generated.

UnsetBad2DStereo

This flag is used with the Add2D constant. In certain cases the 2D layout may result in incorrect stereochemistry in the output connection table format. When this flag is set the writer will output the generated coordinates and set the stereochemistry of the offending stereocenters to 'Undefined'.

MCHG

MISO

MRGP

MV30

MDLParity

If this flag is set and *OEMDLHasParity* returns `false` ,then *OEMDLPerceiveParity* function is called to perceive the MDL atom stereo parity fields.

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWriteMDLFile*.

NoParity

If this flag is set, then *OEMDLClearParity* function is called to set all MDL atom stereo parity fields to zero.

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWriteMDLFile*.

CurrentParity

If this flag is set, then the MDL atom stereo parity fields are left intact.

> **Note**
>
> This flag is only interpreted by the high-level file writers (*OEWriteMolecule*) and not in *OEWriteMDLFile*.

MMask

PMask

Combination of *OEOFlavor.SDF.MDLParity*, *OEOFlavor.SDF.NoParity* and *OEOFlavor.SDF.CurrentParity* flags. The flag controls the processing of the MDL stereo parity bit field of each atom.

SuppressImp2ExpENHSTE

This flavor suppresses any implicit to explicit enhanced stereogroup conversions if the output is V3000 format.

SuppressTimestamps

This flavor suppresses the output of timestamps in MDL format header line(s). This can be useful for generation of simple diffable standards for use in tests.

DEFAULT

Combination of *OEOFlavor.SDF.Add2D*, *OEOFlavor.SDF.MDLParity*, *OEOFlavor.SDF.MRGP*, *OEOFlavor.SDF.MISO* and *OEOFlavor.SDF.MCHG* flags.

Default

Same as the *OEOFlavor.SDF.DEFAULT* constant.

OEOFlavor::SLN

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the Tripos model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

Default

OEOFlavor::SMI

The *OEOFlavor.SMI* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OECreateSmiString* function when writing a molecule into a *OEFormat.SMI* format file.

> **Note**
>
> If no aromaticity model is specified for the *OEOFlavor.Generic* namespace for this file format, then the OpenEye model is used by default to perceive aromaticity by the *OEWriteMolecule* high-level writer.

This namespace contains the following constants:

AllBonds

Corresponds to the *OESMILESFlag.AllBonds* constant.

AtomMaps

Corresponds to the *OESMILESFlag.AtomMaps* constant.

AtomStereo

Corresponds to the *OESMILESFlag.AtomStereo* constant.

BondStereo

Corresponds to the *OESMILESFlag.BondStereo* constant.

Canonical

Corresponds to the *OESMILESFlag.Canonical* constant.

DEFAULT

Same as the *OEOFlavor.SMI.Default* constant.

Default

Combination of *OEOFlavor.SMI.AtomMaps*, *OEOFlavor.SMI.AtomStereo*, *OEOFlavor.SMI.BondStereo*, *OEOFlavor.SMI.Canonical*, *OEOFlavor.SMI.Isotopes* and *OEOFlavor.SMI.RGroups* flags.

EnhStereo

Corresponds to the *OESMILESFlag.EnhStereo* constant.

ExtBonds

Corresponds to the *OESMILESFlag.ExtBonds* constant.

Hydrogens

Corresponds to the *OESMILESFlag.Hydrogens* constant.

ImpHCount

Corresponds to the *OESMILESFlag.ImpHCount* constant.

Isotopes

Corresponds to the *OESMILESFlag.Isotopes* constant.

Kekule

Corresponds to the *OESMILESFlag.Kekule* constant.

RGroups

Corresponds to the *OESMILESFlag.RGroups* constant.

SmiMask

SuperAtoms

Corresponds to the *OESMILESFlag.SuperAtoms* constant.

OEOFlavor::USM

This namespace contains constants.

AllBonds

Corresponds to the *OESMILESFlag.AllBonds* constant.

AtomMaps

Corresponds to the *OESMILESFlag.AtomMaps* constant.

AtomStereo

Corresponds to the *OESMILESFlag.AtomStereo* constant.

BondStereo

Corresponds to the *OESMILESFlag.BondStereo* constant.

Canonical

Corresponds to the *OESMILESFlag.Canonical* constant.

DEFAULT

Same as the *OEOFlavor.USM.Default* constant.

Default

Combination of *OEOFlavor.USM.AtomMaps*, *OEOFlavor.USM.RGroups* flags.

EnhStereo

Corresponds to the *OESMILESFlag.EnhStereo* constant.

ExtBonds

Corresponds to the *OESMILESFlag.ExtBonds* constant.

Hydrogens

Corresponds to the *OESMILESFlag.Hydrogens* constant.

ImpHCount

Corresponds to the *OESMILESFlag.ImpHCount* constant.

Isotopes

Corresponds to the *OESMILESFlag.Isotopes* constant.

Kekule

Corresponds to the *OESMILESFlag.Kekule* constant.

RGroups

Corresponds to the *OESMILESFlag.RGroups* constant.

SmiMask

SuperAtoms

Corresponds to the *OESMILESFlag.SuperAtoms* constant.

OEOFlavor::XYZ

The *OEOFlavor.XYZ* namespace encodes symbolic constants used as bit-masks that control the processing performed by the *OEWriteXYZFile* function when writing a molecule into a *OEFormat.XYZ* format file.

This namespace contains the following constants:

Charges

Specifies that the partial charge for each atom should be appended as an additional column.

Symbols

Write the atomic symbol instead of the atomic number.

SinglePrecision

Forces coordinates to be written in single precision with only 5 significant digits.

DoublePrecision

Forces coordinates to be written in double precision with only 18 significant digits.

Default

Equivalent to `Charges | Symbols`. By default, what precision, single or double precision will be introspected from the molecule type.

OEPDBAtomName

The *OEPDBAtomName* namespace encodes symbolic constants representing PDB atom types.

See also

- PDB file format documentation
- [Markley-1998]
- *OEGetPDBAtomIndex* function
- *OEGetPDBAtomName* function

This namespace contains the following constants:

UNK

Unknown atom name.

N

Peptide backbone amide nitrogen.

CA

Peptide backbone alpha carbon.

C

Peptide backbone carbonyl carbon.

O

Peptide backbone carbonyl oxygen.

CB

Peptide sidechain beta carbon (for example *ALA*).

CG

Peptide sidechain gamma carbon (for example *LYS*).

OG

Peptide sidechain gamma oxygen (for example *SER*).

SG

Peptide sidechain gamma sulfur (for example *CYS*).

CG1

Peptide sidechain beta carbon 1 (for example the long arm of *ILE*).

OG1

Peptide sidechain beta oxygen 1 (for example *THR*).

CG2

Peptide sidechain beta carbon 2 (for example the short arm of *ILE*).

CD

Peptide sidechain delta carbon (for example *LYS*).

OD

Peptide sidechain delta oxygen.

SD

Peptide sidechain delta sulfur (for example *MET*).

CD1

Peptide sidechain delta carbon 1 (for example *ILE*).

ND1

Peptide sidechain delta nitrogen 1 (for example *HIS*).

OD1

Peptide sidechain delta oxygen 1 (for example *ASN*).

CD2

Peptide sidechain delta carbon 2 (for example *HIS*).

ND2

Peptide sidechain delta nitrogen 2 (for example *ASN*).

OD2

Peptide sidechain delta oxygen 2 (for example *ASP*).

SE_

Peptide sidechain selenium (for example *MSE*).

CE

Peptide sidechain epsilon carbon (for example *MET*).

NE

Peptide sidechain epsilon nitrogen (for example *ARG*).

OE

Peptide sidechain epsilon oxygen.

CE1

Peptide sidechain epsilon carbon 1 (for example *HIS*).

NE1

Peptide sidechain epsilon nitrogen 1 (for example *TRP*).

OE1

Peptide sidechain epsilon oxygen 1 (for example *GLN*).

CE2

Peptide sidechain epsilon carbon 2 (for example *TRP*).

NE2

Peptide sidechain epsilon nitrogen 2 (for example *GLN*).

OE2

Peptide sidechain epsilon oxygen 2 (for example *GLN*).

CE3

Peptide sidechain epsilon carbon 3 (for example *TRP*).

OE3

Peptide sidechain epsilon oxygen 3.

OE4

Peptide sidechain epsilon oxygen 4.

CZ

Peptide sidechain zeta carbon (for example *TYR*).

NZ

Peptide sidechain zeta nitrogen (for example *LYS*).

CZ2

Peptide sidechain zeta carbon 2 (for example *TRP*).

CZ3

Peptide sidechain zeta carbon 3 (for example *TRP*).

OH

Peptide sidechain eta oxygen (for example *TYR*).

CH1

Peptide sidechain eta carbon 1.

NH1

Peptide sidechain eta nitrogen 1 (for example *ARG*).

CH2

Peptide sidechain eta carbon 2 (for example *TRP*).

NH2

Peptide sidechain eta nitrogen 2 (for example *ARG*).

CH3

Methyl carbon (for example *ACE*).

OXT

Peptide backbone C-terminal oxygen.

P

Nucleic acid backbone phosphate phosphorus.

O1P

Nucleic acid backbone phosphate oxygen 1.

O2P

Nucleic acid backbone phosphate oxygen 2.

O5_

Nucleic acid backbone ribose oxygen 5-prime.

C5_

Nucleic acid backbone ribose carbon 5-prime.

C4_

Nucleic acid backbone ribose carbon 4-prime.

O4_

Nucleic acid backbone ribose oxygen 4-prime.

C3_

Nucleic acid backbone ribose carbon 3-prime.

O3_

Nucleic acid backbone ribose oxygen 3-prime.

C2_

Nucleic acid backbone ribose carbon 2-prime.

O2_

Nucleic acid backbone ribose oxygen 2-prime.

C2A

Methylated nucleic acid base carbon 2A (for example *2MG*).

C1_

Nucleic acid backbone ribose carbon 1-prime.

N9

Nucleic acid base nitrogen 9.

C8

Nucleic acid base carbon 2.

N7

Nucleic acid base nitrogen 7.

C5

Nucleic acid base carbon 5.

C6

Nucleic acid base carbon 6.

O6

Nucleic acid base oxygen 6.

N6

Nucleic acid base nitrogen 6.

N1

Nucleic acid base nitrogen 1.

C2

Nucleic acid base carbon 2.

O2

Nucleic acid base oxygen 2.

N2

Nucleic acid base nitrogen 2.

CM5

Methylated nucleic acid base carbon M5 (for example *5MC*).

N3

Nucleic acid base nitrogen 3.

C3

Nucleic acid base carbon 3.

C4

Nucleic acid base carbon 4.

O4

Nucleic acid base oxygen 4.

N4

Nucleic acid base nitrogen 4.

CM7

Methylated nucleic acid base carbon M7 (for example *7MG*).

C5M

Methylated nucleic acid base carbon 5M (for example methylated *T*).

O3P

Nucleic acid terminal phosphate oxygen 3P (for example *1MA*).

CM1

Methylated nucleic acid base carbon M1 (for example *1MA*).

CM2

Methylated nucleic acid base carbon M2 (for example *2MG*).

C10

Modified nucleic acid base carbon 10 (for example *YG*).

C11

Modified nucleic acid base carbon 11 (for example *YG*).

C12

Modified nucleic acid base carbon 12 (for example *YG*).

C13

Modified nucleic acid base carbon 13 (for example *YG*).

C14

Modified nucleic acid base carbon 14 (for example *YG*).

C15

Modified nucleic acid base carbon 15 (for example *YG*).

C16

Modified nucleic acid base carbon 16 (for example *YG*).

O17

Modified nucleic acid base oxygen 17 (for example *YG*).

O18

Modified nucleic acid base oxygen 18 (for example *YG*).

C19

Modified nucleic acid base carbon 19 (for example *YG*).

N20

Modified nucleic acid base nitrogen 20 (for example *YG*).

C21

Modified nucleic acid base carbon 21 (for example *YG*).

O22

Modified nucleic acid base oxygen 22 (for example *YG*).

O23

Modified nucleic acid base oxygen 23 (for example *YG*).

C24

Modified nucleic acid base carbon 24 (for example *YG*).

Cprev

Refers to previous peptide backbone carbonyl carbon.

Nnext

Refers to next peptide backbone amide nitrogen.

CAnext

Refers to next peptide backbone alpha carbon.

OEPerceived

The *OEPerceived* namespace encodes symbolic constants representing whether or not various atom, bond or molecular properties have been perceived of a molecule.

See also

- *OEMolBase.HasPerceived* method
- *OEMolBase.SetPerceived* method

Aromaticity

The *OEPerceived.Aromaticity* flag is set by the *OEAssignAromaticFlags* function.

AtomStereo

The *OEPerceived.AtomStereo* flag is set by the following functions:

- *OE3DToAtomStereo*
- *OEMDLStereoFromBondStereo*
- *OEMDLStereoFromParity*

BondStereo

The *OEPerceived.BondStereo* flag is set by the *OE3DToBondStereo* function.

Chains

Currently not used by *OEChem*.

Chiral

The *OEPerceived.Chiral* flag is set by the *OEPerceiveChiral* function.

Energy

The *OEPerceived.Energy* flag is set by the *OEMolBase.SetEnergy* method.

Hybridization

The *OEPerceived.Hybridization* flag is set by the *OEAssignHybridization* function.

PartialCharges

Currently not used by *OEChem*.

RingAtomsAndBonds

The *OEPerceived.RingAtomsAndBonds* flag is set by the *OEFindRingAtomsAndBonds* function.

SecondaryStructure

The *OEPerceived.SecondaryStructure* flag is set by the *OEReadPDBFile* function.

OEPreserveResInfo

The *OEPreserveResInfo* namespace encodes symbolic constants used as bit-masks to indicate which fields in the *OEResidue* class are preserved during an *OEPerceiveResidues* function call.

This namespace contains the following constants:

All

AlternateLocation

AtomName

ChainID

BFactor

Default

FragmentNumber

HetAtom

InsertCode

None

Occupancy

ResidueName

ResidueNumber

SerialNumber

OEProperty

This namespace contains constants.

Undefined

Undefined

Aromatic

Aromatic

Ring

Ring

Donor

Donor

Acceptor

Acceptor

Both

Both

Chiral

Chiral

Exo

Exo

Endo

Endo

StereoDefined

StereoDefined

Visit

Visit

Deleted

Deleted

Single

Single

Double

Double

Triple

Triple

Closure

Closure

Rotor

Rotor

RingAtomsAndBonds

RingAtomsAndBonds

ClosureBonds

ClosureBonds

Chirality

Chirality

Hybridization

Hybridization

Kekule

Kekule

Degree

Degree

Valence

Valence

PhCorrected

PhCorrected

AtomTypes

AtomTypes

Residue

Residue

BondOrders

BondOrders

HydrogenMode

HydrogenMode

DeletedAtoms

DeletedAtoms

DeletedBonds

DeletedBonds

AtomTypeString

AtomTypeString

BondTypeString

BondTypeString

Isotope

Isotope

PartialCharge

PartialCharge

AtomTypeInt

AtomTypeInt

AtomName

AtomName

X

X

Y

Y

Z

Z

MatchOrder

MatchOrder

OEBase

OEBase

Rxn

Rxn

RxnRole

RxnRole

AtomMemberPred

AtomMemberPred

BondMemberPred

BondMemberPred

BondStereo

BondStereo

Component

Component

AroModel

AroModel

PartialChargeModel

PartialChargeModel

Dimension

Dimension

Perceived

Perceived

MapIdx

MapIdx

SymClass

SymClass

All

All

Radius

Radius

Anisou

Anisou

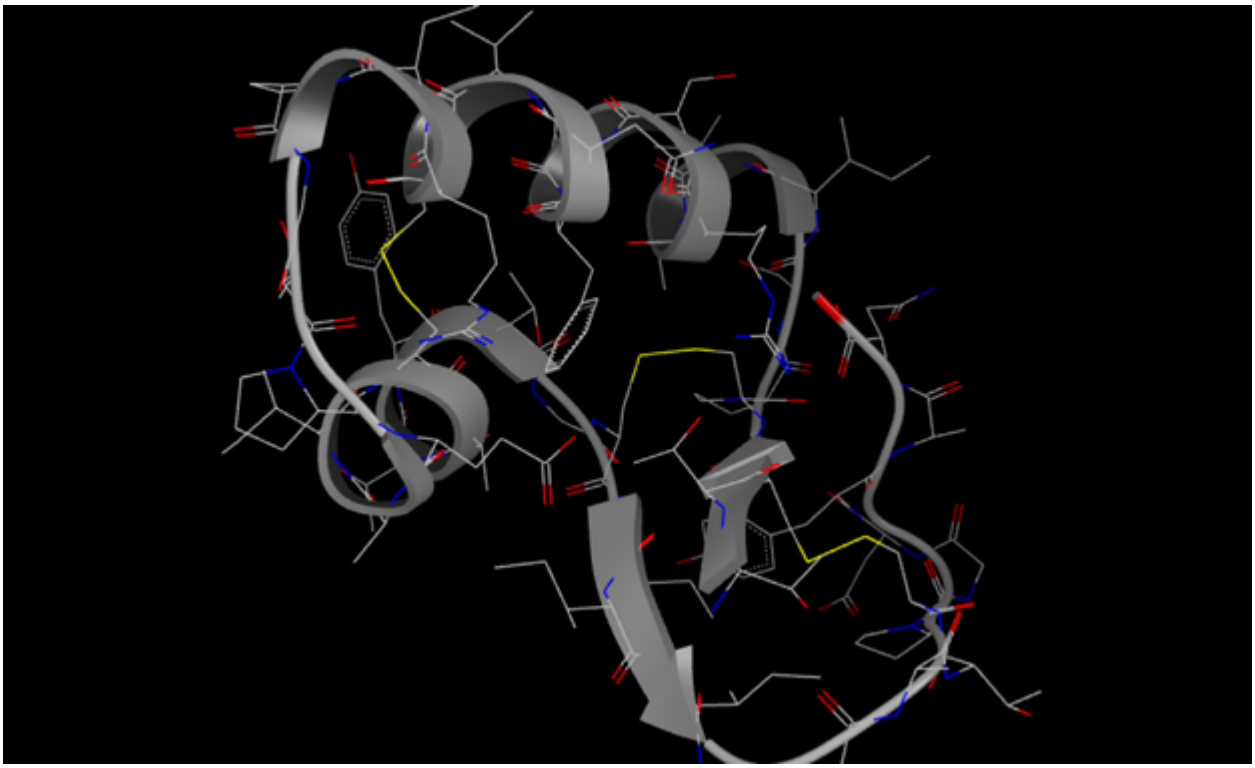
Monomer

Used by *OEHelmToMol* function to attach monomer (*OEMonomerData*) into a molecule.

OEProteinColorScheme

This namespace contains constants.

AtomColor



BEGIN

END

OEProteinStyle

The OEProteinStyle namespace encodes symbolic constants representing the different protein styles that can be stored in an *OE3DMolStyle*. The constants can be used as an argument for the *OE3DMolStyle.SetProteinStyle* function in *OE3DMolStyle*.

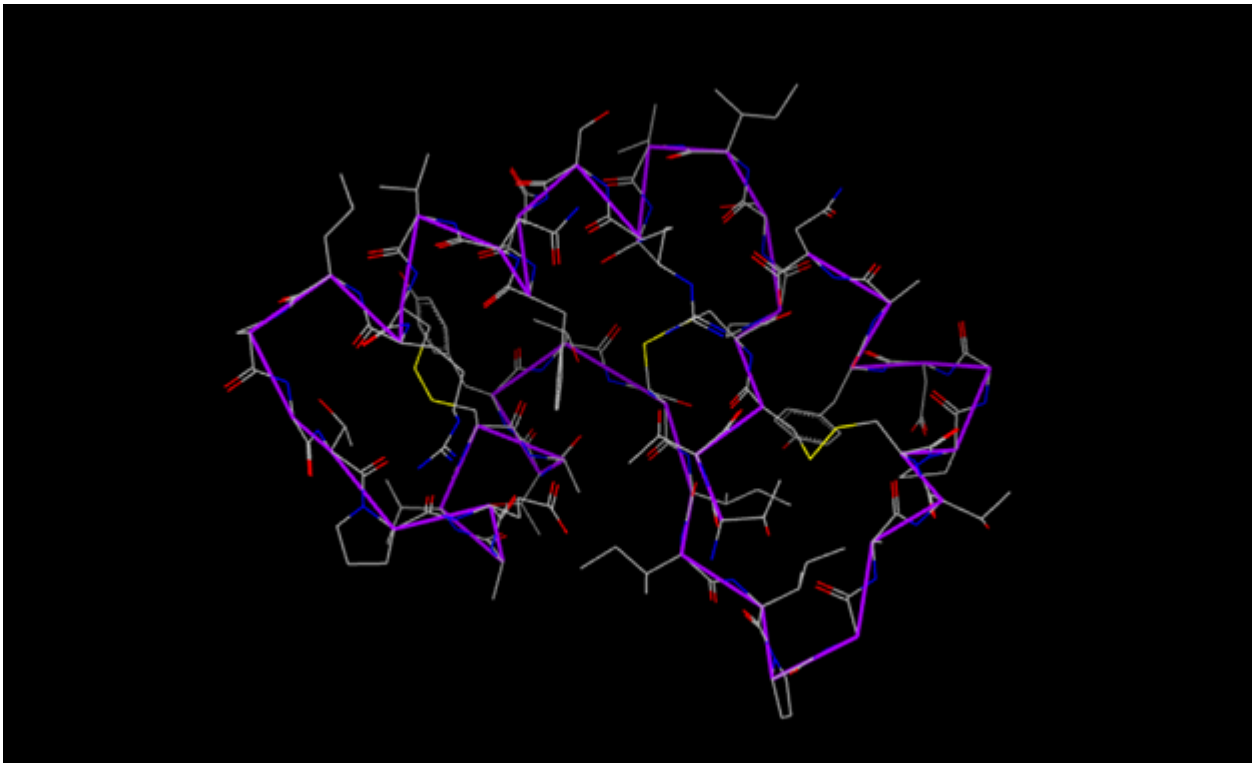
This namespace contains constants.

BEGIN

Backbone

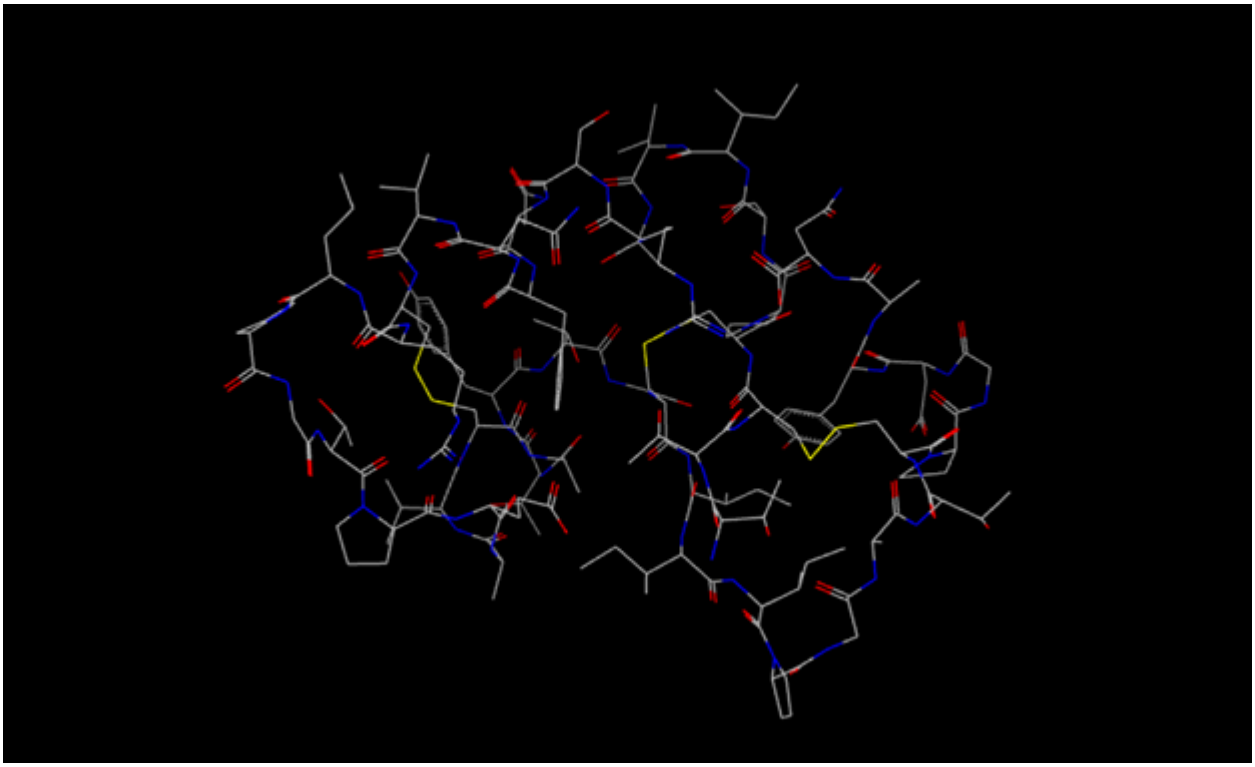


CAlpha

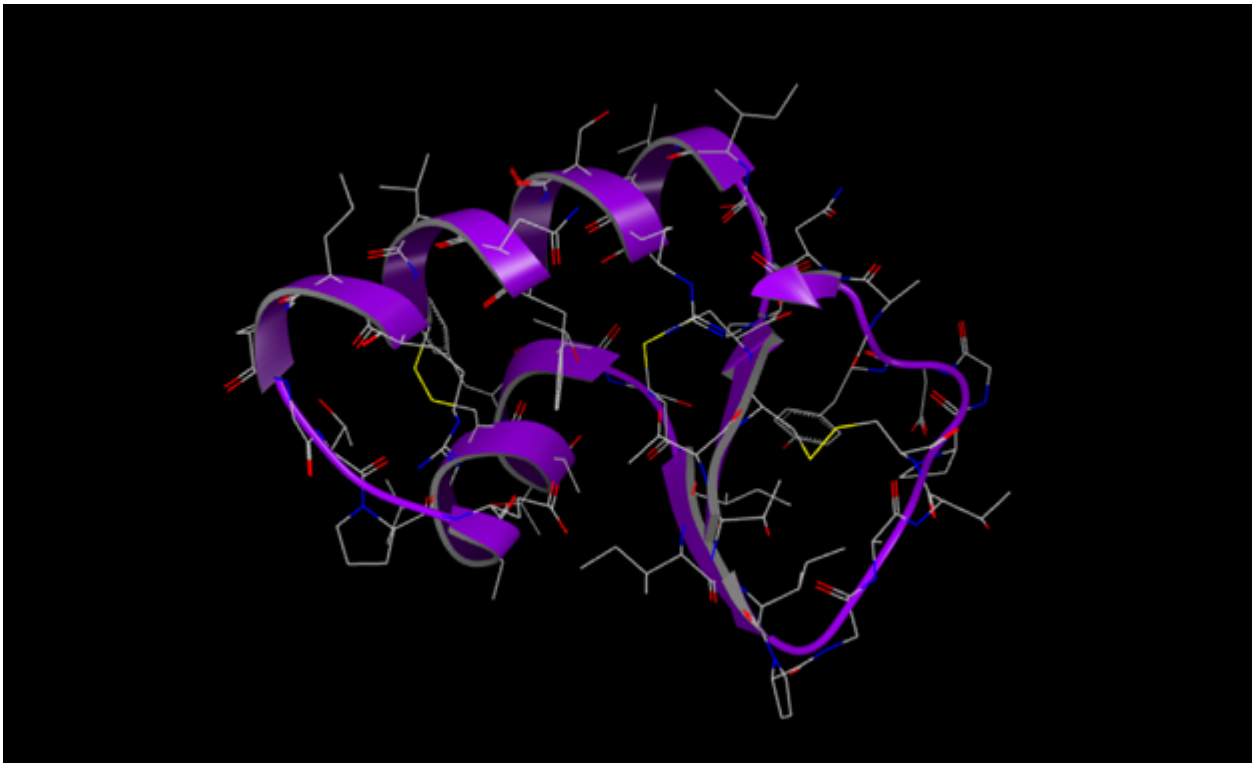


END

Off



Ribbons



OEQMolType

The *OEQMolType* namespace encodes symbolic constants representing specific *OEQMolBase* class implementation types.

See also

- *OENewQMolBase* function

This namespace contains the following constants:

Undefined

OEDefault

MaxType

OERadiiType

This namespace contains the following constants used as options to *OEAssignRadii*.

See also

- *Atom Radius Functions* section
- *OEZap7Version*
- *OEZap9Version*

BondiHVdw

Bondi van der Waals radii, with polar hydrogen radii set to 0.95 Å.

BondiVdw

Assign Bondi van der Waals radii using *OEAssignBondiVdWRadii*.

Covalent

Assign covalent bonding radii using *OEAssignCovalentRadii*.

Default

When used as `molRadiiType`, equates to *OERadiiType.BondiVdw* When used as `ionRadiiType`, equates to whatever is specified for `molRadiiType`.

Delphi

Assign DelPhi radii using *OEAssignDelphiRadii*.

Delphi_Robust

DelPhi radii where the radii of unsupported atoms are assigned radii from *OERadiiType.BondiVdw*.

HonigIonicCavity

Assign Honig effective ionic cavity radii using *OEAssignHonigIonicCavityRadii*.

HonigIonicCavity_Robust

Honig ionic cavity radii where the radii of unsupported atoms are assigned radii from *OERadiiType.Delphi* or *OERadiiType.BondiVdw*, in that order.

Max

Total number of atomic radii constants.

None

Atomic radii are all set to 0.0.

PaulingVdw

Assign Pauling van der Waals radii using *OEAssignPaulingVdWRadii*.

PaulingVdw_Robust

Pauling van der Waals radii where the radii of unsupported atoms are assigned radii from *OERadiiType.Delphi* or *OERadiiType.BondiVdw*, in that order.

Zap7

Equivalent to *OERadiiType.Zap7_CarbonylOxygen*.

Zap7_CarbonylOxygen

Assign *Zap7* radii using *OEAssignZap7Radii* with the parameter *OEZap7Version.CarbonylOxygen*.

Zap7_DoublyBondedOxygen

Assign *Zap7* radii using *OEAssignZap7Radii* with the parameter *OEZap7Version.DoublyBondedOxygen*.

Zap9

Equivalent to *OERadiiType.Zap9_JCAMD2010*.

Zap9_JCAMD2010

Assign *Zap9* radii using *OEAssignZap9Radii* with the parameter *OEZap9Version.JCAMD2010*.

Zap9_JMC2008

Assign *Zap9* radii using *OEAssignZap9Radii* with the parameter *OEZap9Version.JMC2008*.

OEResidueIndex

The *OEResidueIndex* namespace encodes symbolic constants representing residue indices returned by the *OEGetResidueIndex* function(s).

If the biopolymer residue name is not recognized, the *OEGetResidueIndex* function returns either a zero or a unsigned integer value other than the values listed below.

> **Note**
>
> As future releases of **OEChem** may add to this namespace, code should not test for zero, but instead check against the relevant values and treat all other values as 'unrecognized'.

This namespace contains the following constants:

ALA

ARG

ASN

ASP

CYS

GLN

GLU

GLY

HIS

ILE

LEU

LYS

MET

PHE

PRO

SER

THR

TRP

TYR

VAL

UNK

ASX

GLX

ASH

CYX

CYH

CYM

GLH

HID

HIE

HIP

LYN

TYM

A

C

G

I

T

U

DA

DC

DG

DI

DU

_1MA

_2MG

_5MC

_5MU

_7MG

H2U

M2G

OMC

OMG

PSU

YG

ABA

ACE

CGU

CME

CSD

FOR

HSE

HYP

HYL

LYZ

MEN

MLE

MLY

MSE

ORN

PCA

PTR

SAR

SEP

TAU

TPO

LIG

MOL

UNL

BR

CL

ADP

AF3

ALF

ATP

BME

DMS

EDO

EOH

FAD

GOL

HEM

HOH

IOD

MOH

NAD

NAG

PEO

PER

PO2

PO4

SO2

SO4

SX

NHE

NME

CIP

CIM

RU

SB

V

MN

MN3

FE

FE2

CO

CO3

NI

_3NI

CU1

CU

Y1

YT3

_4MO

_6MO

EU

EU3

AU

AU3

PT4

PT

IR3

IR

OS

OS4

YB

YB2

_4TI

OERxnRole

The *OERxnRole* namespace encodes symbolic constants representing for the 'role' property of *OEAtomBase* objects This stored property is used to record that 'role' played by the atom in a reaction molecule.

See also

- *OEAtomBase.GetRxnRole* method
- *OEAtomBase.SetRxnRole* method

This namespace contains the following constants:

None

Reactant

Agent

Catalyst

Product

OESLNFlag

This namespace contains constants.

Isotopes

Controls whether isotopic mass is written to the generated SLN strings.

Hydrogens

Controls whether explicit hydrogens are automatically suppressed and written to the SLN string as part of an atom's implicit hydrogen count.

Canonical

Controls whether we attempt to generate canonical SLN strings.

Kekule

Controls whether we generate Kekulé or aromatic SLN strings. The *OECreateSlnString* function uses the currently assigned aromaticity model. Ideally, from compatibility with Tripos generated SLN, the user should assign Tripos aromaticity to the molecule when writing aromatic SLN, calling the function *OEAssignAromaticFlags* with *OEAroModel.Tripos* aromaticity model.

Name

Controls whether we append the SLN name attribute to the end of each SLN string.

FCharge

Controls whether we output the partial charge on each atom.

Coords2D

Coords3D

DEFAULT

The default value, *OESLNFlag.DEFAULT* currently contains the flavor *OESLNFlag.Name*.

OESmartsParseOpts

The *OESmartsParseOpts* namespace encodes symbolic constants used as bit-masks to indicate how to interpret SMARTS and SMIRKS strings.

See also

- *OEParseSmarts* function
- *OEParseSmirks* function

This namespace contains constants.

Default

The combination of the following flags.

- *OESmartsParseOpts.RingConstraint*
- *OESmartsParseOpts.Optimize*

None

Only constraints explicitly specified in the SMARTS and SMIRKS strings are added to the *OEQMolBase* query structure.

Optimize

When the *OESmartsParseOpts.Optimize* option is turned on, the atom and bond expressions are optimized for substructure search.

RingConstraint

Additional ring constraint is added to each ring atom of the generated *OEQMolBase* query molecule.

This can significantly increase the performance of the substructure search. For example, matching the C1CCCCC1 ring against the CCCCCCCC chain will fail much faster, since none of the query ring atoms can be mapped to any of the target chain atom.

OESMILESFlag

This namespace contains constants that controls the generation of SMILES when calling the low-level *OECreateSmiString* function.

Table 35: **The following function calls generate identical SMILES**

OEChem function	calling *OECreateSmiString* with flags from the *OESMILESFlag* namespace
OECreateAbsSmiStr	*AtomMaps* | *RGroups*
OECreateCanSmiStr	*DEFAULT* identical to *Canonical* | *AtomMaps* | *RGroups*
OECreateIsoSmiStr	*ISOMERIC* identical to *Isotopes* | *AtomStereo* | *BondStereo* | *Canonical* | *AtomMaps* | *RGroups*
OEMolToSmiles	*ISOMERIC* identical to *Isotopes* | *AtomStereo* | *BondStereo* | *Canonical* | *AtomMaps* | *RGroups*

[1] *OEMolToSmiles* is a high-level function that perceives necessary properties before generating canonical isomeric SMILES

AllBonds

Controls whether all bonds are written explicitly to the generated SMILES string. By default, aromatic bonds and most single bonds will be omitted since the semantics are unambiguous.

This option is convenient for using SMILES as a generic graph representation language and for simple SMILES processors which do not perceive aromaticity.

For example, will generate `c:1:c:c:c:c:c:1` for benzene rather than `c1ccccc1`.

AtomMaps

Controls whether atom map indices are written to the generated SMILES string. *OEChem* allows the use of atom map indices on molecules other than reactions, *e.g.* `[CH4:1]` SMILES.

AtomStereo

Controls whether the specified chirality at tetrahedral atomic stereocenters should be written to the generated SMILES string.

BondStereo

Controls whether the specified cis/trans chirality at bond stereocenters should be written to the generated SMILES string.

Canonical

Controls whether to generate canonical SMILES strings. The canonical SMILES strings generated by *OEChem* are not canonicalized by the same algorithm used by Daylight Chemical Information Systems, and hence some molecules will be assigned a different unique representation by each toolkit. Indeed, the algorithms used both by OpenEye and Daylight evolve and improve over time, so SMILES should always be recanonicalized by the same version of a toolkit, before comparing them for equivalence.

EnhStereo

Controls whether explicit enhanced stereogroup information is exported as a CXSMILES formatted appendix following the generated SMILES string. The CXSMILES appendix is whitespace delimited and begins and ends with `|`.

ExtBonds

Controls whether atoms with atomic number zero (as determined by the *OEAtomBase.GetAtomicNum* method), and a non-zero map index (as determined by the *OEAtomBase.GetMapIdx* method) should be generated the external bond &1 notation. In this notation, the integer value following the & corresponds to the atom's map index. When this flag isn't set, such atoms are written in the Daylight convention `[*:1]`.

Hydrogens

Controls whether explicit hydrogens are included in the SMILES string. If not set, explicit hydrogens are included in each atom's implicit hydrogen count.

ImpHCount

Controls whether implicit hydrogen count is written to the generated SMILES strings. For example, to generate `[cH]1[cH][cH][cH][cH][cH]1` for benzene rather than `c1ccccc1`.

Isotopes

Controls whether isotopic mass is written to the generated SMILES strings.

Kekule

Controls whether to generate Kekulé or aromatic SMILES strings. The *OECreateSmiString* function uses the currently assigned aromaticity model. There are known portability problems with different SMILES parsers having varying degrees of ability in parsing aromatic systems, hence writing Kekulé SMILES or using a simple model of aromaticity (see the function *OEAssignAromaticFlags*) can be used to improve interoperability with non-OpenEye software.

RGroups

Controls whether atoms with atomic number zero (as determined by the *OEAtomBase.GetAtomicNum* method), and a non-zero map index (as determined by the *OEAtomBase.GetMapIdx* method) should be displayed using the `[R1]` notation. In this notation, the integer value following the `R` corresponds to the atom's map index. When this flag isn't set, such atoms are written in the Daylight convention `[*:1]`.

SuperAtoms

Controls whether atoms with atomic number zero (as determined by the *OEAtomBase.GetAtomicNum* method), and a non-empty name string property (as determined by the *OEAtomBase.GetName*) should be displayed in SMILES *superatom* notation, rather than just `[*]`. For example, this can be used to generate the SMILES `[Asp][Gly][Lys]`.

DEFAULT

The combination of *OESMILESFlag.RGroups*, *OESMILESFlag.AtomMaps* and *OESMILESFlag.Canonical* flavors, that is used to generate canonical SMILES

ISOMERIC

The combination of *OESMILESFlag.RGroups*, *OESMILESFlag.AtomMaps*, *OESMILESFlag.Canonical*, *OESMILESFlag.Isotopes*, *OESMILESFlag.AtomStereo* and *OESMILESFlag.BondStereo* flavors, that is used to generate canonical isomeric SMILES.

OEStyleGridType

This namespace contains constants.

BEGIN

END

Electrostatic

OESubSearchDatabaseType

> **Attention**
>
> This API is currently available in C++ and Python.

This namespace contains constants representing how search data (molecules and pre-generated screens) are stored and searched in an *OESubSearchDatabase* object.

This namespace contains the following constants.

Default

The default type is `OESubSearchDatabaseType.InMemoryMultiThreaded`.

InMemoryMultiThreaded

When an *OESubSearchDatabase* is initialized with the `OESubSearchDatabaseType.InMemoryMultiThreaded` type, the search data, both molecules and pre-generated screens, are pre-loaded into memory.

This mode is memory intensive, but it provides the **fastest** way to search a dataset when utilizing all available processors on the machine.

MolDatabaseMultiThreaded

When an *OESubSearchDatabase* is initialized with the `OESubSearchDatabaseType.MolDatabaseMultiThreaded` type, **only** the pre-generated screens are pre-loaded into memory. During the search, if a molecule can not be rejected based on its screen, then it has to be loaded from disk on-the-fly to perform atom-by-atom validation.

This mode is not as fast as `OESubSearchDatabaseType.InMemoryMultiThreaded` mode, however, it requires significantly less memory, allowing to searches on larger datasets. The performance loss is highly dependent on the query and its screening performance. If a significantly large number of hits can be pre-screened, searching only a couple thousands molecules can be done very fast even with the added I/O cost of loading molecules on the fly.

> **Note**
>
> In both modes, searches can be executed in single or multi-treaded way based on the parameter (maximum number of processors) that was determined when the database was constructed.

> **See also**
>
> - *OESubSearchDatabase* class

> **Code Example**
>
> - *Perform SMARTS search on substructure database file* example

OESubSearchScreenType

> **Attention**
>
> This API is currently available in C++ and Python.

The *OESubSearchScreen* objects are typed bit-vectors. The type of a screen determines what kind of queries in mind the screen was designed.

In **OEChem TK**, a query molecule (*OEQMolBase*) can be constructed in three different ways.

Screen Type	Description	Initialize	See Also
OESubSearchType.MDL	query read from MDL query file	*OEReadMDLQueryFile*	*Substructure Search with MDL Queries*
OESubSearchType.Molecule	query built from molecule	*BuildExpressions*	
OESubSearchType.SMARTS	query built from SMARTS pattern	*OEParseSmarts*	*SMARTS Pattern Matching*

The expression trees of the queries that are initialized from different sources can be significantly different. For example, the "language" of SMARTS is the most expressive one allowing complex logical operations and recursive patterns. A SMARTS such as `[NX3;H2,H1;!$(NC=O)]` (primary or secondary amine that is not in amide) defines a substructure that can not be expressed using MDL query features. This also makes queries built from SMARTS the most complex ones. The other major difference between SMARTS and MDL queries is the way how they are expected to match aromaticity. For these reasons, **OEChem TK** provides three different screen types especially designed for the three different ways that are available to built query molecules using **OEChem TK**.

> **Note**
>
> For performance reason, **OEChem TK** does not support screening-based fast substructure search with explicit hydrogens. The *OEMakeSubSearchTargetScreen* function automatically suppresses explicit hydrogens of the molecule prior to the screen generation. The *OEMakeSubSearchQueryScreen* function will return `false` if the query molecule has any explicit hydrogen expression.

Default

The default screen type is *OESubSearchScreenType.MDL*.

MDL

Molecule

Note

- It is highly recommended to use the *OEExprOpts.DefaultAtoms* and *OEExprOpts.DefaultBonds* options when building the query molecule with the *BuildExpressions*. When using other option combinations, the built query molecule might not be suitable for the *OESubSearchScreenType.Molecule* screen type.
- If the input molecule has any explicit hydrogens, the *OESuppressHydrogens* function should be called prior to calling *BuildExpressions*.

SMARTS

Hint

In case of a query molecule of unknown source, the *OEIsComplementaryScreenType* function can be used to check whether the query molecule is suited for a specific screen type.

See also

- *OEMakeSubSearchQueryScreen* and *OEMakeSubSearchTargetScreen* functions

Code Example

- *Generate database for fast substructure search* example
- *Perform SMARTS search on substructure database file* example

OESubSearchStatus

Attention

This API is currently available in C++ and Python.

This namespace contains constants representing the various status of the substructure search (*OESubSearchResult.GetSearchStatus*).

Canceled

This constant indicates that the substructure search was canceled by invoking *OESubSearchResult.Cancel*.

Finished

This constant indicates that the substructure search was finished successfully.

InProgress

This constant indicates that the substructure search is in progress.

Uninitialized

This constant indicates that the substructure search can not be initialized. For example, attempting a search on an invalid *OESubSearchDatabase* object.

See also

- `OESubSearchDatabase.Search` method
- `OESubSearchStatusToName` function

Code Example

- *Perform SMARTS search on substructure database file* example

OESurfaceColorScheme

The OESurfaceColorScheme namespace encodes symbolic constants representing the different surface color schemes. The constants can be used as an argument when constructing a *OEMolStyleColorer* which can be used as an argument for the *SetSurfaceColorer* method in *OE3DMolStyle*.

This namespace contains constants.

AtomColor

BEGIN

Concavity

Curvature

Distance

END

Electrostatic

GenericData

Grid

HBondPotential

Hydrophobicity

Solid

SurfacePotential

OESurfaceStyle

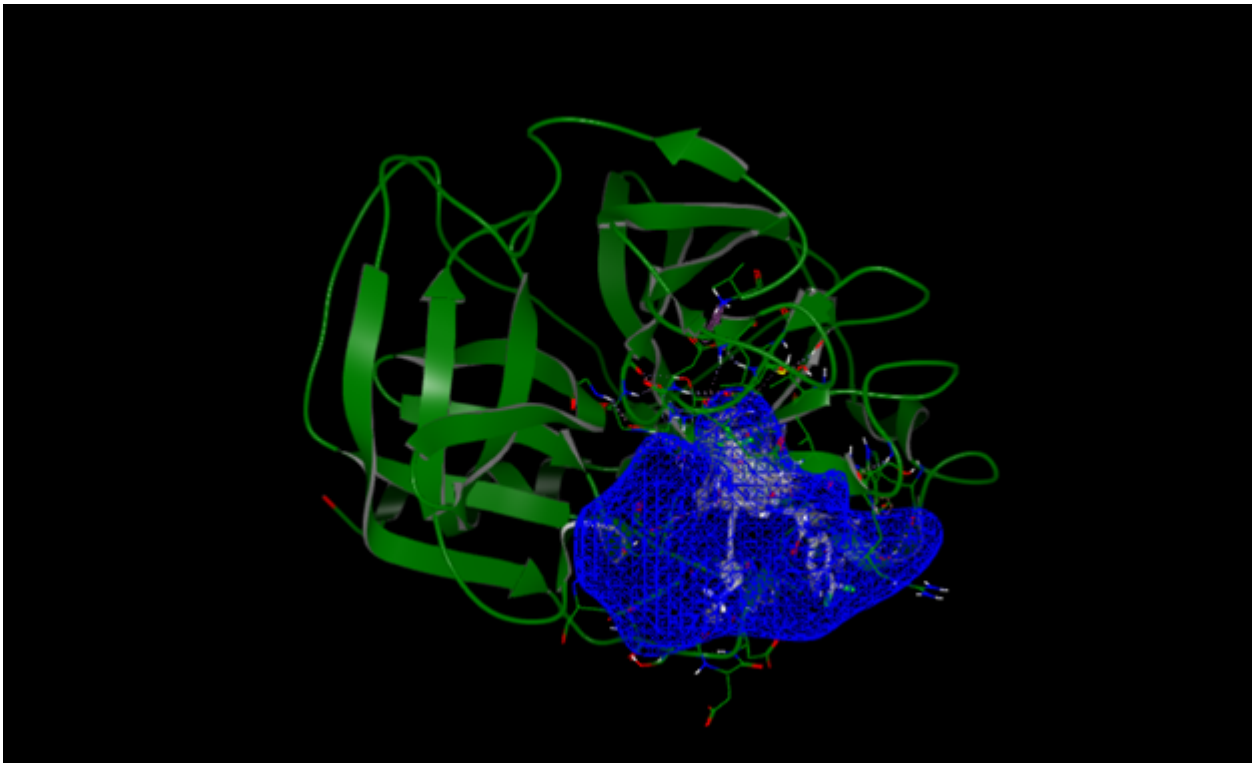
The OESurfaceStyle namespace encodes symbolic constants representing the different surface styles for the *OE3DMolStyle*. The constants can be used as an argument for the *OE3DMolStyle.SetSurfaceStyle* function in *OE3DMolStyle*.

This namespace contains constants.

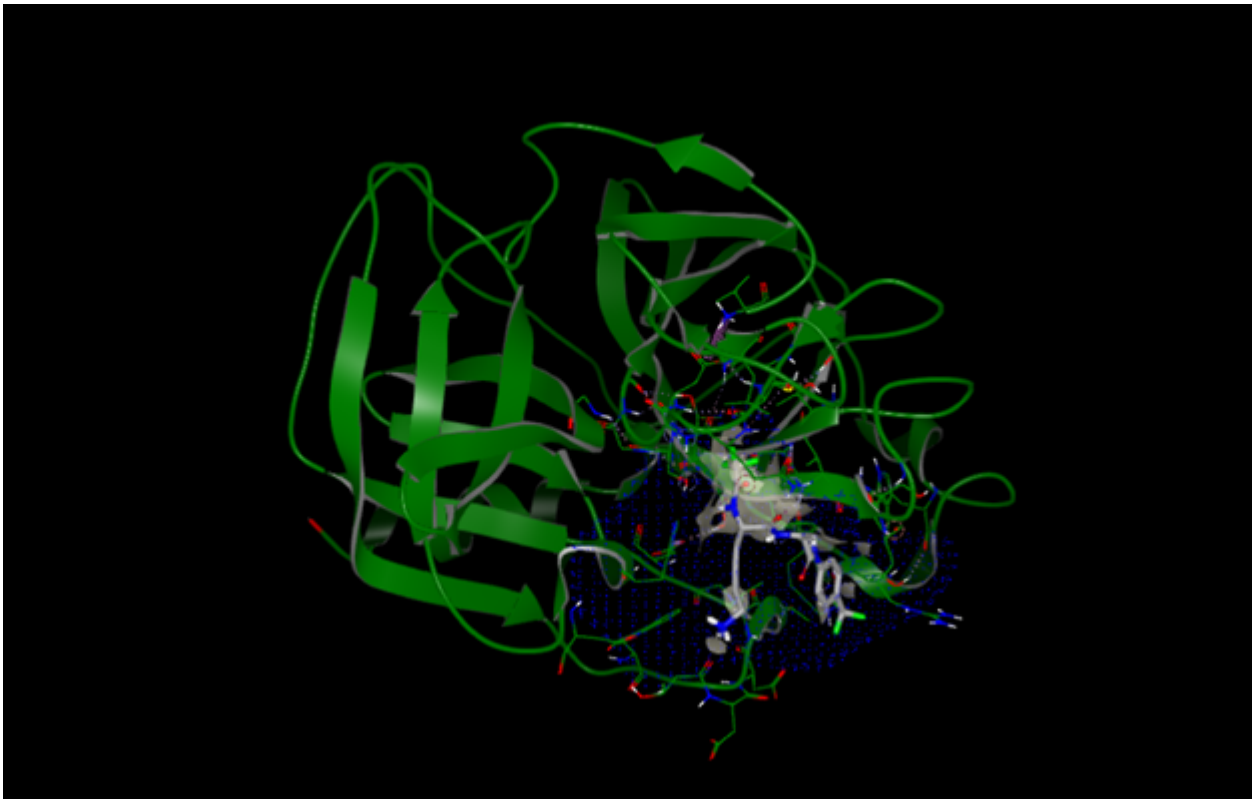
BEGIN

END

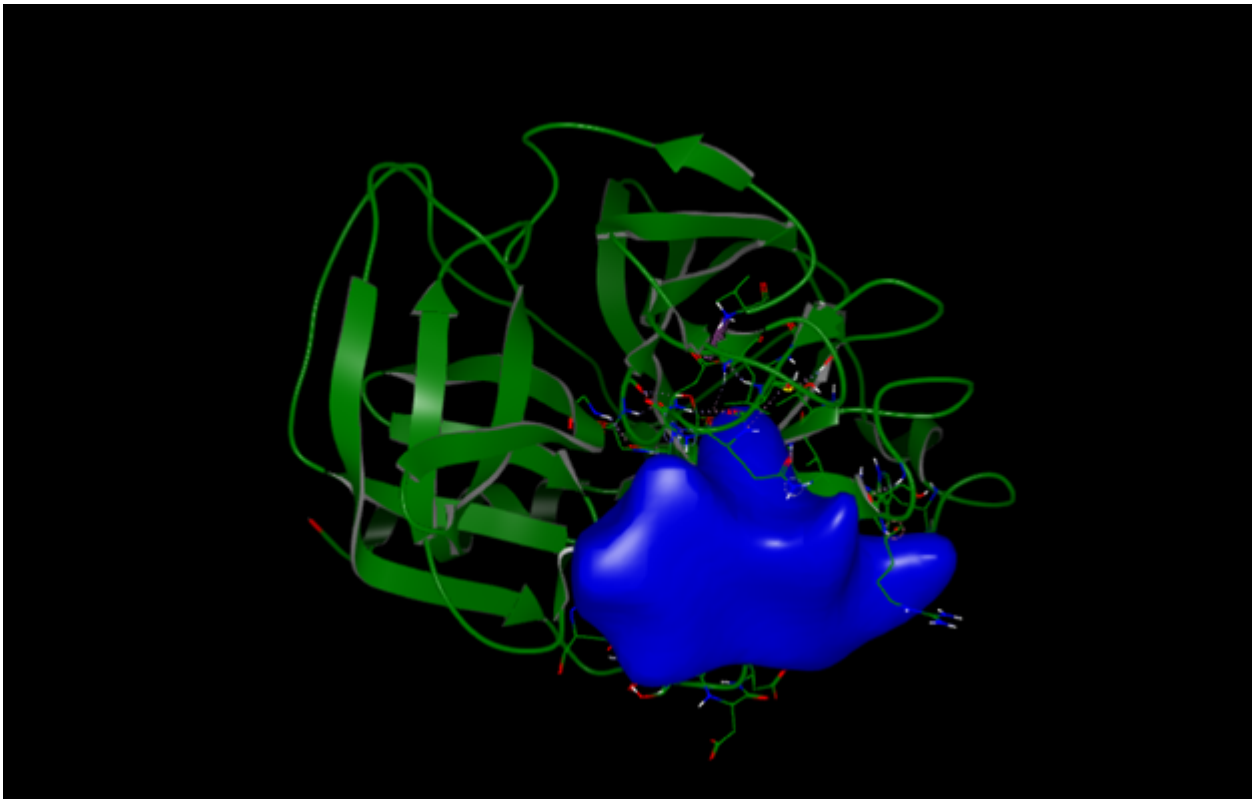
Mesh



Points



Solid



OESurfaceType

The OESurfaceType namespace encodes symbolic constants representing the different surface types for the *OE3DMolStyle*. The constants can be used as an argument for the *OE3DMolStyle.SetSurfaceType* function in *OE3DMolStyle*.

This namespace contains constants.

Accessible

BEGIN

END

Molecular

Off

VDW

OETriposType

The *OETriposType* namespace encodes symbolic constants representing Tripos atom types. These values are typically calculated by atom typing, read from a Sybyl mol2 format file and used in writing Sybyl mol2 format files.

This namespace contains the following constants:

Du

Dummy

Al

Aluminium

Br

Bromine

C1

Carbon sp

C2

Carbon sp^2

C3

Carbon sp^3

Car

Carbon aromatic

Ccat

C^+ carbocation

Ca

Calcium

Cl

Chlorine

Cooh

Cobalt (octahedral)

Crth

Chromium (octahedral)

Croh

Chromium (tetrahedral)

Cu

Copper

F

Fluorine

Fe

Iron

H

Hydrogen

I

Iodine

K

Potassium

Li

Lithium

Lp

Lone pair

Mg

Magnesium

Mn

Manganese

Mo

Molybdenum

N1

Nitrogen sp

N2

Nitrogen sp^2

N3

Nitrogen sp^3

N4

Nitrogen sp^3 with positive charge

Nam

Nitrogen in amide

Nar

Nitrogen aromatic

Npl3

Nitrogen trigonal planar

Na

Sodium

O2

Oxygen sp^2

O3

Oxygen sp^3

Oco2

Oxygen in carboxylate

P3

Phosphorous sp^3

S2

Sulfur sp^2

S3

Sulfur sp^3

Se

Selenium

So

Sulfur in sulfoxide

So2

Sulfur in sulfone

Si

Silicon

Sn

Tin

Zn

Zinc

MAXTYPE

OEUncolorStrategy

The *OEUncolorStrategy* namespace encodes symbolic constants representing types of actions for molecule uncoloring activities.

See also

- *OEUncolorMol* function

This namespace contains the following constants:

Default

The default uncolor action is *OEUncolorStrategy.RemoveDimension* and *OEUncolorStrategy.BemisMurcko*.

UncolorNone

This mode requests no changes be made to the structure.

ConvertAtomTypeToC

Convert all non-hydrogen atom types to carbon.

ConvertBondTypeToSingle

Convert all bond types to single bonds.

RemoveAtomStereo

Remove any atom stereo parity values as well as any stereo bond marks, also implies RemoveGroupStereo.

RemoveBondStereo

Remove any cis/tran bond parity values as well as any double either marks.

RemoveGroupStereo

Removes all MDL stereogroups (''abs", ''or", and ''and" groups).

RemoveAtomProperties

Remove the following properties from the atoms: charge, aromaticity, isotope and map index values.

RemoveDimension

Clears the coordinates, resets stereo perception flags and sets the coordinate dimension to `0`.

BemisMurcko

This mode embodies the following actions: *OEUncolorStrategy.ConvertAtomTypeToC*, *OEUncolorStrategy.ConvertBondTypeToSingle*, *OEUncolorStrategy.RemoveAtomStereo*, *OEUncolorStrategy.RemoveBondStereo*, *OEUncolorStrategy.RemoveAtomProperties*

OEUniMolecularRxnFixValence

This namespace contains constants representing various supported valence fixup actions after a successfully applied reaction transformation.

All

This constant requests that all atoms in the transformed products be checked for valence issues.

Altered

This constant requests that only atoms modified by the transformation in the transformed products be checked for valence issues.

DEFAULT

Default

This constant requests the default action valence correction setting of *OEUniMolecularRxnFixValence.Explicit*.

Explicit

This constant requests that no atoms in the transformed products be checked for valence issues. It is assumed that the reaction transformation itself provides expicit valence corrections as part of the transform.

OEWriteMolReturnCode

The *OEWriteMolReturnCode* namespace encodes symbolic constants representing error codes returned by the high-level write functions *OEWriteMolecule* and *OEWriteConstMolecule*.

This namespace contains the following constants:

Success

This constant indicates that the molecule has been written out successfully.

FormatNotWritable

This constant indicates that an unsupported output format for write was attempted. See *OEIsWriteable*.

OEBWriteFailure

This constant indicates a general error occurred when serializing the structure to *OEFormat.OEB*.

TooManyAtoms

This constant indicate that the molecule exceeds the limits for the specified format, for example: >99999 atoms for *OEFormat.PDB*.

NoConformer

This constant indicates that writing a multi-conformer molecule has failed due to a missing conformation.

InvalidSDData

This constant indicates that invalid SD data is present on the structure which would result in a noncompliant output *OEFormat.SDF* format. An attempt was made to alter the output data to ensure compliance with strict readers.

MolInvalidForFileFormatSpec

This constant indicates that a molecule cannot be written to the requested format, because it would violate the format spec. As an example, not all molecules read from MMCIF can be written to the PDB file format. Examples issues are, multi-character chain IDs and residue names exceeding the 3-character limit in PDB format.

OEZap7Version

See also

- *OEAssignZap7Radii* function

This namespace contains the following constants:

Undefined

CarbonylOxygen

Zap7 radii version in which carbonyl oxygens radii parameter is increased with respect to the Bondi radius, while all other oxygens have radii set at Bondi values.

DoublyBondedOxygen

Modified version of Zap7 radii in which in addition to carbonyl oxygens, all other doubly bonded oxygens have increased radii parameter with respect to the Bondi radii (see Table 2 in [Nicholls-2010]).

Default

Same as the *OEZap7Version.CarbonylOxygen* constant.

OEZap9Version

See also

- *OEAssignZap9Radii* function

This namespace contains the following constants:

Undefined

JMC2008

Initial Zap9 radii version described in [Nicholls-2008].

JCAMD2010

Modified version of Zap9 radii from [Nicholls-2010].

Default

Same as the *OEZap9Version.JCAMD2010* constant.

4.1.3 OEChem Functions

OE3DToAtomStereo

```
bool OE3DToAtomStereo(OEMolBase& mol)
```

Set the stereochemistry at the chiral atoms of a molecule, as specified by the molecule's 3D coordinates. The affected atoms are those with a 'chiral' atom property of `true`, as perceived by a call to *OEPerceiveChiral*, or set manually by the user calling the *OEAtomBase.SetChiral* method. If no chirality has been perceived on the molecule, *OEPerceiveChiral* will be called inside this function.

This function will return `false` immediately, if the dimension of the molecule is not found to be 3.

The *OE3DToAtomStereo* function throws a warning and returns `false` at the end, if the geometry of any of the chiral atoms of the given molecule is not tetrahedral *i.e.* if the atom stereo can not be determined from the coordinates.

After stereochemistry is set *SetPerceived* is called with the *OEPerceived.AtomStereo* flag set to `true`. Subsequent calls to *OE3DToAtomStereo* will return `true` immediately as the value of *HasPerceived* is checked on entry. Stereochemistry can be forced to be reperceived by setting the *OEPerceived.AtomStereo* flag to `false`.

OE3DToBondStereo

```
bool OE3DToBondStereo(OEMolBase& mol, unsigned char* bmask = 0)
```

Set the stereochemistry at the chiral bonds of a molecule, as specified by the molecule's 3D (or possibly 2D) coordinates. The affected bonds are those with a 'chiral' bond property of `true`, as discovered by a call to *OEPerceiveChiral*, or set manually by the user calling *OEBondBase.SetChiral* method. If no chirality has been perceived or set on the molecule, *OEPerceiveChiral* will be called inside this function.

After stereochemistry is set *SetPerceived* is called with the *OEPerceived.BondStereo* flag set to `true`. Subsequent calls to *OE3DToBondStereo* will return `true` immediately as the value of *HasPerceived* is checked on entry. Stereochemistry can be forced to be reperceived by setting the *OEPerceived.BondStereo* flag to `false`.

OE3DToInternalStereo

```
bool OE3DToInternalStereo(OEMolBase& mol)
```

Assigns the stereochemistry of molecule from its 3D coordinates. It perceives both the tetrahedral chirality around atomic centers, and the E/Z chirality around double bonds. This function is equivalent to the sequence of calling *OE3DToAtomStereo* and *OE3DToBondStereo*. However *OE3DToBondStereo* will not be called if *OE3DToAtomStereo* returns `false`.

OEAcyclicPath

```
OESystem::OEIterBase<OEAtomBase>* OEAcyclicPath(const OEAtomBase* bgn,
                                                const OEAtomBase* end)
```

Returns an iterator over the atoms in order connecting `bgn` to `end` through the molecular graph without traversing rings. Both `bgn` and `end` will be returned in the iterator. Therefore, `bgn` will always be the first atom in the iterator and `end` will always be the last atom in the iterator if an acyclic path through the molecule exists. If no path exists, only `bgn` will be returned in the iterator.

OEAddCustomFASTAResidue

```
bool OEAddCustomFASTAResidue(const char* code, const char* smiles)
```

Add a custom residue to the **OEChem TK** FASTA parser. The `code` is the string surrounded in square brackets. The `smiles` is the structure of the un-natural nucleic acid to add to the internal dictionary. Note, the hydroxyl from the background will be automatically stripped and removed by this function.

In order for these residues to be used, the *OEIFlavor.FASTA.CustomResidues* flavor must be specified. The following code snippet demonstrates how to register a custom D-isoleucine with the FASTA reader named with `[dI]`. It also demonstrates using the *OEIFlavor.FASTA.EmbeddedSMILES* flavor to define a tether that can cyclize the peptide.

> **Warning**
>
> This is an experimental API that may change in the future.

OEAddExplicitHydrogens

```
bool OEAddExplicitHydrogens(OEMolBase& mol, bool polarOnly = false,
                            bool set3D = true)
bool OEAddExplicitHydrogens(OEMCMolBase& mol, bool polarOnly = false,
                            bool set3D = true)
```

Converts the implicit hydrogens on the atoms of a molecule to explicit hydrogen atoms. Zero or more new atoms are created with atomic number *OEElemNo.H*, and new bonds are created to connect these to the parent atom with a single bond. This function also resets the implicit hydrogen count to zero.

polarOnly
: If `true`, this function only add explicit hydrogens to polar atoms, *i.e.* when *OEAtomBase.IsPolar* returns `true`.

set3D
: If `true`, the default, and the molecule is 3D, then the *OESet3DHydrogenGeom* function is automatically called on any hydrogen atom in the molecule that has the same coordinates as its parent heavy atom.

  If `false`, then the sprouted hydrogens are initially given the same coordinates as their parent heavy atom.

Hint

In case of a 2D molecule, the coordinates of the explicit hydrogens can be assigned by calling the *OESet2DHydrogenGeom* function.

```
bool OEAddExplicitHydrogens(OEMolBase& mol, OEAtomBase* hvyatom)
```

This function creates a new *OEAtomBase* for every implicit hydrogen on 'hvyatom', as given by *OEAtomBase.GetImplicitHCount*. The new hydrogen atoms are given the same coordinates as their parent, 'hvyatom'. This function currently always returns `true`. To set the 2D or 3D geometry of these atoms, call the *OESet2DHydrogenGeom* or *OESet3DHydrogenGeom* functions, respectively.

See also

- *OESuppressHydrogens* function
- *OESet2DHydrogenGeom* function
- *OESet3DHydrogenGeom* function

OEAddMMCIFData

```
bool OEAddMMCIFData(OEMolBase& mol, const std::string& value)
```

Adds mmCIF data to a molecule. The input is not validated, but is added to the mmCIF metadata block.

See also

- *OESetMMCIFData* function to set (replace) MMCIF data
- *mmCIF Metadata Manipulation* section

Note

Unicode characters are not supported neither in the data tag nor in the data value.

OEAddMols

```
bool OEAddMols(OEMolBase& dst, const OEMolBase& src,
               OEAtomBase** amap = (OEAtomBase **) 0,
               OEBondBase** bmap = (OEBondBase **) 0)
bool OEAddMols(OEMolBase& dst, const OEMolBase& src,
               const char* delim,
               OEAtomBase** amap = (OEAtomBase **) 0,
               OEBondBase** bmap = (OEBondBase **) 0)
```

Adds the one molecule to another molecule.

dst
: The destination molecule.

src
The molecule that will be added to the destination molecule.

delim
The delimiter used to concatenate the title of the two molecules. The empty string (*e.g.* "") is a valid argument, and causes a direct concatenation of the two titles. If `(char *)0` is passed as the argument, no concatenation of titles occurs. If not specified _ character will be used as to concatenate the titles together.

amap
This parameter can be used to obtain mappings between the atoms in the original `src` molecule and the newly created atoms in the `dst` molecule. It is assumed that argument passed to `amap` is array of atom pointers of at least *src.GetMaxAtomIdx()* in size.

The newly created atom corresponding to a source atom can be obtained by atom index lookup into the `amap` array.

bmap
This parameter can be used to obtain mappings between the bonds in the original `src` molecule and the newly created bonds in the `dst` molecule. It is assumed that argument passed to `bmap` is array of bond pointers of at least *src.GetMaxBondIdx()* in size.

The newly created atom corresponding to a source atom can be obtained by atom index lookup into the `bmap` array.

Note

OEAddMols will not generate bonds between the molecules added.

See also

- Example program *CatMols.cs*

OEAddMolStyles

```
void OEAddMolStyles(OEChem::OEMolBase& lhs,
                    const OEChem::OEMolBase& rhs,
                    OEChem::OEAtomBase** aa, OEChem::OEBondBase **)
```

Merges styles from two molecules, usually utilized when concatenating molecules. *OEMolBase* &rhs must have a style because *OEMolBase* &lhs will inherit the style from *OEMolBase* &rhs.

OEAddPDBData

```
bool OEAddPDBData(OEMolBase& mol, const OESDDataPair& dp)
bool OEAddPDBData(OEMolBase& mol, const std::string& tag,
                  const std::string& value)
```

Adds a PDB tag and value data pair to a molecule. Both the tag and the value must be a string. If an item with the same tag already exists, another item is added to the molecule.

See also

- *OESetPDBData* function to set (replace) PDB data
- *PDB Tagged Data Manipulation* section

Note

Unicode characters are not supported neither in the data tag nor in the data value.

OEAddSDData

```
bool OEAddSDData(OEMolBase& mol, const OESDDataPair& dp)
bool OEAddSDData(OEMolBase& mol, const std::string& tag,
                 const std::string& value)
```

Adds an SD tag and value data pair to a molecule. Both the tag and the value must be a string. If an item with the same tag already exists, another item is added to the molecule.

See also

- *OESetSDData* function to set (replace) SD data
- *SD Tagged Data Manipulation* section

Note

Unicode characters are not supported neither in the data tag nor in the data value.

OEApplyStyleChange

```
OE3DMolStyle OEApplyStyleChange(const OE3DMolStyle& initialStyle,
                                const OE3DMolStyle& deltaStyle)
```

Returns a new *OE3DMolStyle*, which includes all of the specified options in the initial style that do not overlap with the delta style and any new additions from the delta style.

OEAssignAromaticFlags

```
void OEAssignAromaticFlags(OEMolBase& mol,
                           unsigned int aromodel = OEAroModelOpenEye,
                           bool clearflags = true,
                           unsigned int maxpath = 0,
                           bool prune = false)
```

Perceives the aromatic atoms and bonds of a molecule. The assigned atom and bond aromaticity can be accessed by *OEAtomBase.IsAromatic* and *OEBondBase.IsAromatic* methods, respectively.

model

The parameter that specifies that aromaticity model that are perceived by the *OEAssignAromaticFlags*

function. The default aromaticity 'model' is *OEAroModel.OpenEye*. Other predefined aromaticity models provided by **OEChem TK** include *OEAroModel.Daylight*, *OEAroModel.Tripos*, *OEAroModel.MMFF* and *OEAroModel.MDL* that represent the Daylight, Tripos, MMFF and MDL definitions, respectively.

clearflags
The parameter that specifies whether to clear the aromaticity flags by calling *OEClearAromaticFlags* prior to re-perceiving the aromaticity. Newly created molecules that have not had their aromaticity assigned yet can specify `false`, for a very small performance advantage.

maxpath
The parameter that specifies the maximum path length to consider an aromatic cycle, or zero (the default) to specify no upper bound on aromatic cycle length. Some formal models of aromaticity use the value six, limiting aromaticity to six membered rings like benzene or pyridine.

prune
The parameter that specifies whether or not to run a post-processing step to consider rings with exo-double bonds as not aromatic. This is also required by some formal models of aromaticity.

See also

- *OEAroModel* constant namespace
- *OEClearAromaticFlags* to remove atom and bond aromaticity properties
- *Aromaticity Perception* chapter
- Example program *CanSmi.cs*

OEAssignBondiVdWRadii

```
void OEAssignBondiVdWRadii(OEMolBase& mol)
```

Assigns radii to all atoms in the given *OEMolBase* using calls to *OEGetBondiVdWRadius*. The assigned radii of an *OEAtomBase* can be accessed by *OEAtomBase.GetRadius* method.

See also

- *Atom Radius Functions* section

OEAssignCovalentRadii

```
void OEAssignCovalentRadii(OEMolBase& mol)
```

Assigns radii to all atoms in the given *OEMolBase* using calls to *OEGetCovalentRadius*. The assigned radii of an *OEAtomBase* can be accessed by *OEAtomBase.GetRadius* method.

See also

- *Atom Radius Functions* section

OEAssignDelphiRadii

```
void OEAssignDelphiRadii(OEMolBase& mol)
```

Assigns radii to all atoms in the given *OEMolBase* using calls to *OEGetDelphiRadius*. The assigned radii of an *OEAtomBase* can be accessed by *OEAtomBase.GetRadius* method.

See also

- *Atom Radius Functions* section

OEAssignFormalCharges

```
void OEAssignFormalCharges(OEAtomBase* atm)
```

Sets the formal charge property of an atom, based upon a **OpenEye valence model**. This function assumes that the bond orders and implicit hydrogen counts have been set on a molecule. For example, this function will place a single positive charge on a neutral four-valent nitrogen. Note that this function only modifies neutral atoms, with zero formal charge. This preserves any formal charges previously assigned, but requires the charge to be reset to zero in order to force a reassignment.

The assigned formal charge can be accessed by the *OEAtomBase.GetFormalCharge* method

```
void OEAssignFormalCharges(OEMolBase& mol)
```

Sets the formal charge property of each atom in the molecule by by calling *OEAssignFormalCharges(OEAtomBase *)* function.

The assigned formal charge can be accessed by the *OEAtomBase.GetFormalCharge* method.

See also

- *OpenEye Valence Model* section
- *OEAssignImplicitHydrogens* function

OEAssignHonigIonicCavityRadii

```
void OEAssignHonigIonicCavityRadii(OEMolBase& mol)
```

Assigns radii to all atoms in the given *OEMolBase* using calls to *OEGetHonigIonicCavityRadius*. The assigned radii of an *OEAtomBase* can be accessed by *OEAtomBase.GetRadius* method.

See also

- *Atom Radius Functions* section

OEAssignHybridization

```
bool OEAssignHybridization(OEMolBase &)
```

Assigns hybridization to all atoms in the given *OEMolBase* using calls to *OEGetHybridization* function that predicts the electronic hybridization/geometry of each atom from simple connectivity

This function updates the hybridization property of each atom in the molecule (see namespace *OEHybridization*). The assigned hybridization value can be accessed by the *GetHyb* method of *OEAtomBase*.

Warning

The predicted atom hybridization depends on the perceived aromaticity.

1. If the aromaticity is not perceived prior to calling the *OEAssignHybridization* function, then *OEAssignAromaticFlags* is automatically called to perceive the OpenEye aromaticity model. For more details about aromaticity models, see *Aromaticity Perception*)
2. Otherwise, the already perceived aromaticity information is used.

OEAssignImplicitHydrogens

```
bool OEAssignImplicitHydrogens(OEAtomBase* atm)
```

Sets the implicit hydrogen count property of an atom based upon a **OpenEye valence model**. This function assumes that the formal charges may be incorrect or haven't been assigned yet. This function should be followed with a call to the function *OEAssignFormalCharges*. This function calls the *OEAtomBase.SetImplicitHCount* method on the atom with the value returned by *OEDefaultImplicitHCount*.

This function returns `true` if the implicit hydrogen count of the atom is changed.

The assigned implicit hydrogen count can be accessed by *OEAtomBase.GetImplicitHCount* method.

```
bool OEAssignImplicitHydrogens(OEMolBase& mol)
```

Sets the implicit hydrogen count property of each atom in the molecule by calling *OEAssignImplicitHydrogens(OEAtomBase *)*.

This function returns `true` if the implicit hydrogen count of any atom in the molecule is changed. It returns `false` if none of the atoms in the molecule has the implicit hydrogen count changed.

The assigned implicit hydrogen count can be accessed by *OEAtomBase.GetImplicitHCount* method.

Note

If the charge state is correctly set on each atom, you should call the *OEAssignMDLHydrogens* function instead.

See also

- *OpenEye Valence Model* section
- *OEDefaultImplicitHCount* function
- *OEAssignFormalCharges* function

OEAssignMDLHydrogens

```
bool OEAssignMDLHydrogens(OEMolBase& mol)
```

Sets the implicit hydrogen count property of each atom in the molecule based upon the **MDL valence model** by calling *OEAssignMDLHydrogens(OEAtomBase*)* with the value returned by *OEDefaultImplicitHCount*. This function returns `true` if the implicit hydrogen count of any atom in the molecule is changed. It returns `false` if none of the atoms in the molecule has the implicit hydrogen count changed. The assigned implicit hydrogen count can be accessed by *OEAtomBase.GetImplicitHCount* method.

```
bool OEAssignMDLHydrogens(OEAtomBase* atm)
```

Sets the implicit hydrogen count property of an atom based upon the **MDL valence model**. This function assumes that the formal charges are correctly set. This function calls the *OEAtomBase.SetImplicitHCount* method on each atom with the value returned by *OEDefaultMDLHCount*. This function returns `false` if the implicit hydrogen count of the atom is not changed. If the charge state of each atom has not been assigned, you should call the *OEAssignImplicitHydrogens* and *OEAssignFormalCharges* functions instead.

See also

- *MDL Valence Model* section
- *OEDefaultMDLHCount* function
- *OEMDLGetValence* function
- *OEIsValidMDLAtomValence* predicate

OEAssignPaulingVdWRadii

```
void OEAssignPaulingVdWRadii(OEMolBase& mol)
```

Assigns radii to all atoms in the given *OEMolBase* using calls to *OEGetPaulingVdWRadius*. The assigned radii of an *OEAtomBase* can be accessed by *OEAtomBase.GetRadius* method.

See also

- *Atom Radius Functions* section

OEAssignRadii

```
bool OEAssignRadii(OEChem::OEMolBase& mol,
                   unsigned molRadiiType = OERadiiType::Default,
                   unsigned ionRadiiType = OERadiiType::Default)
```

Assigns radii to all atoms in the given *OEMolBase* using calls to specific radii functions based on the values of `molRadiiType` and `ionRadiiType`, drawn from the *OERadiiType* namespace. Ions in a molecular complex can be assigned radii distinct from other species by specifying a value for `ionRadiiType` (for example *OERadiiType.HonigIonicCavity_Robust*).

The assigned radii of an *OEAtomBase* can be accessed by *OEAtomBase.GetRadius* method.

See also

- *Atom Radius Functions* section
- *OEGetRadiiTypeName*

OEAssignResidueNumbers

```
void OEAssignResidueNumbers(OEMolBase& mol)
```

Assigns sequential residue numbers to the residues of a molecule, as currently ordered in the *OEMolBase*. The residue number is stored in *OEResidue* that is associated with each *OEAtomBase* via the *OEProperty.Residue* property.

See also

- *OEAtomGetResidue*

OEAssignSerialNumbers

```
void OEAssignSerialNumbers(OEMolBase& mol)
```

The function assigns sequential atom serial numbers to the atoms of a molecule, as currently ordered in the *OEMolBase*. The atom serial number is stored in *OEResidue* that is associated with each *OEAtomBase* via the *OEProperty.Residue* property.

See also

- *OEAtomGetResidue*

OEAssignZap7Radii

```
void OEAssignZap7Radii(OEMolBase& mol,
                       unsigned int version = OEZap7Version::Default)
```

Assigns atomic radii from the ZAP7 set ([Nicholls-2010]) to all atoms of the passed molecule `mol`. ZAP7 set has been optimized to be used together with high quality atomic partial charges obtained with DFT calculations using large basis set, for example aug-cc-pVTZ, and a surface-area coefficient of 5.7 cal per square Å. The assigned radii of an *OEAtomBase* can be accessed by the *OEAtomBase.GetRadius* method.

See also

- *OEZap7Version*
- *Atom Radius Functions* section

OEAssignZap9Radii

```
void OEAssignZap9Radii(OEMolBase& mol,
                       unsigned int version = OEZap9Version::Default)
```

Assigns radii to all atoms in the given *OEMolBase* using atomic radii defined initially in [Nicholls-2008]. These radii include minor atom-typing for oxygen and nitrogen and are not a simple atomic number mapping. In the later paper [Nicholls-2010] the same modification of radii for some types of doubly bonded oxygen to P,S, and N has been introduced, and is now made a default set of Zap9 radii. These radii are suitable for use in calculating transfer energies when used with a surface-area coefficient of 6.3cal per square Å. The assigned radii of an *OEAtomBase* can be accessed by the *OEAtomBase.GetRadius* method.

See also

- *OEZap9Version*
- *Atom Radius Functions* section

OEAtomGetMDLParity

```
unsigned int OEAtomGetMDLParity(const OEAtomBase* atm)
```

This function is used to retrieve the 'MDL Parity' value associated with an atom. The interpretation of this unsigned integer is described in the MDL file format documentation, and will appear in the appropriate atom parity column of the atom connection tables in MDL file formats. This value may be set using the *OEAtomSetMDLParity* function.

OEAtomGetResidue

```
const OEResidue& OEAtomGetResidue(const OEAtomBase* atm)
```

This function is used to retrieve the *OEResidue* associated with an *OEAtomBase*. The *OEResidue* class is used to store information used in biopolymer processing, *e.g.* proteins and nucleic acids, on a per-atom basis.

See also

- *OEResidue* class
- *OEHasResidue* function
- *OEAtomSetResidue* function

OEAtomGetSmallestRingSize

```
unsigned int OEAtomGetSmallestRingSize(const OEAtomBase* atm)
unsigned int OEAtomGetSmallestRingSize(const OEAtomBase& atm)
```

Determines the size of the smallest ring that an atom is in. If the atom is not in a ring, *i.e.* *OEAtomBase.IsInRing* returns `false`, the value zero is returned.

Warning

It is not necessary to perform the ring perception by calling *OEFindRingAtomsAndBonds* function prior to calling *OEAtomGetSmallestRingSize* function.

See also

- *Ring Perception* chapter

OEAtomIsInAromaticRingSize

```
bool OEAtomIsInAromaticRingSize(const OEAtomBase& atm,
                                unsigned int size)
bool OEAtomIsInAromaticRingSize(const OEAtomBase* atm,
                                unsigned int size)
```

Determines whether the given atom is in an aromatic cycle of the specified size. If the atom is not aromatic, *i.e.* *OEAtomBase.IsAromatic* returns `false`, then this function also returns `false`.

Warning

It is necessary to call *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags* functions prior to calling calling *OEAtomIsInAromaticRingSize* function.

See also

- *Ring Perception* chapter

OEAtomIsInRingSize

```
bool OEAtomIsInRingSize(const OEAtomBase* atm, unsigned int size)
bool OEAtomIsInRingSize(const OEAtomBase& atm, unsigned int size)
```

Determines whether the given atom is in a cycle of the specified size. If the atom is not in a ring, *i.e.* *OEAtomBase.IsInRing* returns `false`, then this function also returns `false`. This function also returns `false` for all values of size less than three.

Warning

It is necessary to perform ring perception by calling *OEFindRingAtomsAndBonds* function prior to calling *OEAtomIsInRingSize* function.

See also

- *Ring Perception* chapter

OEAtomSetMDLParity

```
bool OEAtomSetMDLParity(OEAtomBase* atm, unsigned int)
```

This function may be used to set the 'MDL parity' value associated with an atom. The interpretation of this unsigned integer is described in the MDL file format documentation, and will appear in the appropriate atom parity column of the atom connection tables in MDL file formats. This value may be retrieved using the *OEAtomGetMDLParity* function.

Typically, these values is either read from MDL connection tables or set by *OEChem* via the *OEMDLPerceiveParity* and *OEMDLClearParity* functions.

OEAtomSetResidue

```
bool OEAtomSetResidue(OEAtomBase* atm, const OEResidue& res)
```

This function is used to set the *OEResidue* associated with an *OEAtomBase*. The *OEResidue* class is used to store information used in biopolymer processing, *e.g.* proteins and nucleic acids, on a per-atom basis.

See also

- *OEResidue* class
- *OEHasResidue* function
- *OEAtomGetResidue* function

OEBAtomIntTypeOpts

```
const char* OEBAtomIntTypeOpts()
```

Specifies that atom and bond int types should be read and written from the OEB file format.

OEBDefaultOpts

```
const char* OEBDefaultOpts()
```

The default set of options used by *oemolistream* and *oemolostream*. The default is that rotor-offset-compression is not preserved; as well as that atom and bond int types are not preserved.

OEBGZipCompressOpts

```
const char* OEBGZipCompressOpts()
```

Specifies that the OEB format should gzip compress individual payloads of data.

OEBondGetSmallestRingSize

```
unsigned int OEBondGetSmallestRingSize(const OEBondBase& bnd)
unsigned int OEBondGetSmallestRingSize(const OEBondBase* bnd)
```

Determines the size of the smallest ring that a bond is in. If the bond is not in a ring, *i.e.* *OEBondBase.IsInRing* returns `false`, the value zero is returned.

Warning

It is necessary to perform ring perception by calling *OEFindRingAtomsAndBonds* function prior to calling *OEBondGetSmallestRingSize* function.

See also

- *Ring Perception* chapter

OEBondIsInAromaticRingSize

```
bool OEBondIsInAromaticRingSize(const OEBondBase* bnd,
                                unsigned int size)
bool OEBondIsInAromaticRingSize(const OEBondBase& bnd,
                                unsigned int size)
```

Determines whether the given bond is in an aromatic cycle of the specified size. If the bond is not aromatic, *i.e.* *OEBondBase.IsAromatic* returns `false`, then this function also returns `false`.

Warning

It is necessary to call *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags* functions prior to calling *OEBondIsInAromaticRingSize* function.

See also

- *Ring Perception* chapter

OEBondIsInRingSize

```
bool OEBondIsInRingSize(const OEBondBase* bnd, unsigned int size)
bool OEBondIsInRingSize(const OEBondBase& bnd, unsigned int size)
```

Determines whether the given bond is in a cycle of the specified size. If the bond is not in a ring, *i.e.* *OEBondBase.IsInRing* returns `false`, then this function also returns `false`. This function also returns `false` for all values for size less than three.

Warning

It is necessary to perform ring perception by calling *OEFindRingAtomsAndBonds* function prior to calling *OEBondIsInRingSize* function.

See also

- *Ring Perception* chapter

OEBRotCompressOpts

```
const char* OEBRotCompressOpts()
```

Specifies that while reading a molecule the data needed to maintain a smaller file size should be preserved on the molecule for later use during writing.

OEBuildMDLQueryExpressions

```
bool OEBuildMDLQueryExpressions(OEQMolBase& qmol,
                                const OEMolBase& mol,
                                unsigned int opts = OEMDLQueryOpts::Default)
```

See also

- *Substructure Search with MDL Queries* chapter

OECalcCartesianCoord

```
void OECalcCartesianCoord(float* a, float* b, float* c, float* z,
                          float* x)
```

Given the Cartesian coordinates of the three reference atoms, in the arrays 'a', 'b' and 'c', convert the internal coordinate specified by 'z', to a Cartesian coordinate in the buffer pointed to by 'x'.

OECalcInternalCoord

```
void OECalcInternalCoord(float* a, float* b, float* c, float* x, float* z)
```

Given the Cartesian coordinates of the three reference atoms, in the arrays 'a', 'b' and 'c', convert the Cartesian coordinate specified by 'x', to an internal coordinate in the buffer pointed to by 'z'.

OECalcPMI

```
bool OECalcPMI(double* pmi, OEMolBase& mol)
bool OECalcPMI(double* pmi, const OEMolBase& mol)
bool OECalcPMI(double* pmi, const OEConfBase& mol)
bool OECalcPMI(double* pmi, const OEMCMolBase& mol)
```

Calculates the PMI (principal moment of inertia) of the given molecule. Units of the PMI calculated are amu $\times$ A^2, where amu stands for Atomic Mass Unit and A for Angstrom. If calling the OEMCMolBase overload, the PMI calculated corresponds to the active conformer.

Returns true if calculation is successful, false otherwise. The function returns false if the values for the PMI are all 0, or if the molecule is not centered before calling the overloads with a const molecule argument.

If using any of the constant molecule overloads of this function, you must center the molecule before calling the overload, otherwise the function will return an incorrect or invalid value. See the function *OECenter* for how to center a molecule.

pmi
: This pointer is populated by the PMI of the given molecule in each dimension, X, Y, and Z.

mol
: This is the molecule for which the PMI is calculated. It must have a set radius and set mass for the calculation to be valid.

Warning

These functions only work correctly with a molecule that has been centered using the function *OECenter*. If the molecule has not been centered, the function will return false.

See also

- *OEGetRadiusOfGyration* function to calculate radius of gyration.

The following code snippet demonstrates how to calculate the PMI and radius of gyration for a molecule.

```
using System;
using OpenEye.OEChem;

public class CalcPMI
{
    public static int Main(string[] args)
    {
        foreach (string inp_filename in args)
        {
            OEGraphMol mol = new OEGraphMol();
            oemolistream ifs = new oemolistream(inp_filename);
            double[] pmi = new double[3];
            double[] rGyr = new double[3];
            while (OEChem.OEReadMolecule(ifs, mol))
            {
                OEChem.OECalcPMI(pmi, mol);
                Console.WriteLine("PMI");
                foreach (double val in pmi)
                {
                    Console.Write(val + " ");
                }
                Console.WriteLine();
                OEChem.OEGetRadiusOfGyration(rGyr, pmi, mol);
                Console.WriteLine("Radius of Gyration");
                foreach (double val in rGyr)
```

(continues on next page)

(continued from previous page)

```
                {
                    Console.Write(val + " ");
                }
                Console.WriteLine();
            }
        }
        return 0;
    }
}
```

OECalculateMolecularWeight

```
double OECalculateMolecularWeight(const OEMolBase& mol,
                                  bool isotopic = false)
```

Calculates the molecular weight of the given molecule. By default, all atoms are assumed their average atomic weight, as returned by the *OEGetAverageWeight* function.

isotopic
: This parameter determines whether to use the *OEGetIsotopicWeight* function instead of *OEGetAverageWeight*. Atoms must have a specified (non-zero) isotopic mass, as returned by *OEAtomBase.GetIsotope*

See also

- Example program *SizeFilter.cs*
- Example program *MolStats.cs*

OECanonicalOrderAtoms

```
void OECanonicalOrderAtoms(OEMolBase& mol)
```

Reorders the atoms of the molecule into canonical order. This function uses the *OEMolBase.OrderAtoms* method to change the order in which the iterator returned by *OEMolBase.GetAtoms* traverses the atoms of a molecule. This function does not affect the atom indices.

Note

OECanonicalOrderAtoms does not provide the exactly the same canonicalization that is used to generate canonical isomeric SMILES. The canonical atom order of SMILES generation (with various flavors) can be accessed by calling the *OEGetSmiStringOrder* function.

See also

- Example program *CanSmi.cs*

OECanonicalOrderBonds

```
void OECanonicalOrderBonds(OEMolBase& mol)
```

Reorders the bonds of a molecule canonically with respect to the current atom order. This function uses the *OEMolBase.OrderBonds* method to change the order in which the iterator returned by *OEMolBase.GetBonds* and *OEAtomBase.GetBonds* traverses the bonds of a molecule. This function does not affect the bond indices.

See also

- Example program *CanSmi.cs*

OECenter

```
void OECenter(OEMolBase& mol, double* t = 0)
void OECenter(OEMCMolBase& mcmol, double* t = 0)
void OECenter(OEConfBase& conf, double* t = 0)
```

The *OECenter* function moves the molecule so that its center of mass is located at the Cartesian position origin. An optional array 't' can be passed in, which will be filled with the vector which describes the translation to move the molecule back to its original position. For the *OEMolBase* and *OEConfBase* overloads, the 't' array, if present, is assumed to be at least of length three. For the *OEMCMolBase* overload, the 't' array is assumed to be at least of length *OEMCMolBase.GetMaxConfIdx* `*3`. In this case, the vector to move and conformer back to its original position would be located at `t[OEConfBase::GetIdx()*3]`.

mol
: The molecule to center.

conf
: The conformer to center.

mcmol
: The multi-conformer molecule to center.

t
: The translation required to move molecule back to original position.

OECheckAtomValence

```
bool OECheckAtomValence(const OEAtomBase* atom)
```

Checks the atom's environment (specifically its atomic number, formal charge and the bond orders of the bonds it participates in) to determine if it is a valid atom valence whe consider the **OpenEye valence model**. Returns `true` if the atom is valid.

See also

- section *OpenEye Valence Model*

- *OECheckAtomValences* function
- *OEIsValidAtomValence* predicate

OECheckAtomValences

```
bool OECheckAtomValences(const OEMolBase& mol)
```

Loops through the atoms of the molecule and calls *OECheckAtomValence* to check if every atom is valid considering the **OpenEye valence model**. Returns `true` if all atoms are valid.

See also

- section *OpenEye Valence Model*
- *OECheckAtomValence* function
- *OEIsValidAtomValence* predicate

OEChemGetArch

```
const char* OEChemGetArch()
```

OEChemGetLicensee

```
bool OEChemGetLicensee(std::string& licensee)
```

OEChemGetPlatform

```
const char* OEChemGetPlatform()
```

Returns the internal build string used by OpenEye, Cadence Molecular Sciences to identify a platform. The format of these strings may change over time, and future distributions may contain different values even when using the same operating system, compiler and processor. For example, on a `x86_64` Red Hat Enterprise Linux box this would return `redhat-RHEL5-g++4.1-x64`.

See also

- Example program *OEChemInfo.cs*

OEChemGetRelease

```
const char* OEChemGetRelease()
```

Returns the release name of the *OEChem* library being used. This returns a value similar to `1.0` for production versions of the library, and `1.0 debug` for the checking version of the library.

See also

- *OEChemGetVersion*
- Example program *OEChemInfo.cs*

OEChemGetSite

```
bool OEChemGetSite(std::string& site)
```

OEChemGetVersion

```
unsigned int OEChemGetVersion()
```

Returns the version number of the library being used. This is an unsigned integer value indicating the date on which the library was built, for example `20020903`, for the 3rd of September 2002. This value should be used when reporting problems, and unlike the release string, may be used in comparisons if needed.

See also

- Example program *OEChemInfo.cs*

OEChemIsLicensed

```
bool OEChemIsLicensed(const char* feature = 0,
                      unsigned int* expdate = 0)
```

Determines whether a valid license file is present. This function may be called without a legitimate run-time license to determine whether it is safe to call any **OEChem TK**, **OESystem TK** or **OEPlatform TK** functionality.

The 'features' argument can be used to check for a valid license to **OEChem TK** along with that feature, such as 'python', 'java' or 'clr' (for Csharp). For example, to verify that **OEChem TK** can be used:

```
        if (!OEChem.OEChemIsLicensed("clr"))
            OEChem.OEThrow.Warning("OEChem is not licensed for the C# feature.");
```

The second argument can be used to get the expiration date of the license. This is an array of size three with the date returned as {day, month, year}. Even if the function returns `false` due to an expired license, the 'expdate' will show that expiration date. A value of a zeroes implies that no license or date was found.

```
        uint[] expdate = new uint[3];
        if (OEChem.OEChemIsLicensed("clr", expdate))
        {
            OEChem.OEThrow.Info(String.Format("License expires: day: {0}  month: {1}␣
↪year: {2}",
            expdate[0], expdate[1], expdate[2]));
        }
```

OEClear2DRingDictionary

```
void OEClear2DRingDictionary()
```

Clears the **global** user-defined ring dictionary that is used to generate 2D coordinates when calling the *OEGenerate2DCoordinates* function.

See also

- *OE2DRingDictionary* class
- *OEIsValid2DRingDictionary* function
- *OEInit2DRingDictionary* function

OEClearAromaticFlags

```
void OEClearAromaticFlags(OEMolBase& mol)
```

Clears the aromatic property of all atoms and bonds in the molecule. This function effectively Kekulizes the molecule, if a Kekulé form has already been perceived by the *OEKekulize* function. Note this function does not affect the integer types or the type names of the atom or bond of the molecule.

See also

- *OEAssignAromaticFlags* to determine atom and bond aromaticity properties
- *Aromaticity Perception* chapter

OEClearChiralPerception

```
void OEClearChiralPerception(OEMolBase& mol)
```

Function to clear the chiral perception from a molecule and its atoms and bonds.

See also

- *OEPerceiveChiral* to perceive chiral atoms and bonds

OEClearFormalCharges

```
void OEClearFormalCharges(OEMolBase& mol)
```

Clears the formal charge property of all atoms in a molecule, setting the formal charge on each atom of a molecule to `0`.

OEClearMMCIFData

```
bool OEClearMMCIFData(OEMolBase& mol)
```

Removes all mmCIF data from the molecule.

```
bool OEClearMMCIFData(OEMCMolBase& mol)
```

Removes all mmCIF data from the molecule and from all of its conformers.

See also

- *mmCIF Metadata Manipulation* section

OEClearPartialCharges

```
void OEClearPartialCharges(OEMolBase& mol)
```

Clears the partial charge property of all atoms in a molecule, setting the partial charge on each atom of a molecule to `0.0`.

OEClearPDBData

```
bool OEClearPDBData(OEMolBase& mol)
```

Removes all PDB data from the molecule.

```
bool OEClearPDBData(OEMCMolBase& mol)
```

Removes all PDB data from the molecule and from all of its conformers.

See also

- *OEDeletePDBData* function to remove PDB data with a specific tag
- *PDB Tagged Data Manipulation* section

OEClearResidue

```
void OEClearResidue(OEAtomBase* atom)
```

OEClearResidues

```
void OEClearResidues(OEMolBase& mol)
```

OEClearRotorCompressData

```
bool OEClearRotorCompressData(OEMCMolBase& mol)
```

OEClearSDData

```
bool OEClearSDData(OEMolBase& mol)
```

Removes all SD data from the molecule.

```
bool OEClearSDData(OEMCMolBase& mol)
```

Removes all SD data from the molecule and from all of its conformers.

See also

- *OEDeleteSDData* to remove SD data with a specific tag
- *SD Tagged Data Manipulation* section
- Example program *SDFModProps.cs*

OEClearStyle

```
void OEClearStyle(OEChem::OEMolBase& mol)
void OEClearStyle(OEChem::OEAtomBase& atom)
void OEClearStyle(OESystem::OEContainer& cont)
void OEClearStyle(OESystem::OEIter<OEChem::OEAtomBase>& atoms)
```

Clears the style declared for a molecule and all of its atoms, a certain atom in a molecule, iterator of specified atoms, and an *OEContainer*.

OECopyMMCIFData

```
bool OECopyMMCIFData(OEMolBase& dst, const OEMolBase& src)
```

Copies the entire set of mmCIF metadata from a source molecule to a destination molecule.

See also

- *mmCIF Metadata Manipulation* section

OECopyMol

```
void OECopyMol(OEMolBase& dst, const OEMolBase& src,
               OEAtomBase** atommap = 0, OEBondBase** bondmap = 0)
void OECopyMol(OEMCMolBase& dst, const OEMCMolBase& src,
               OEAtomBase** atommap = 0, OEBondBase** bondmap = 0)
```

Create a copy of the molecule `src` into the molecule `dst`. The mapping of the atoms and bonds between the two copies of the molecule can be retrieved by passing in arrays to the `atommap` and `bondmap` arguments. The size of the `atommap` array should be large enough to store `src.GetMaxAtomIdx()` atoms. The size of the `bondmap` array should be large enough to store `src.GetMaxBondIdx()` atoms. The new atom or bond in `dst` will be stored at the `src` atom or bond's *OEAtomBase.GetIdx* location in the array.

The following code demonstrates how to access the atom associations output by the `OECopyMol` function.

```
using System;
using System.Collections.Generic;
using OpenEye.OEChem;
```

(continues on next page)

(continued from previous page)

```
public class CopyMol
{
    public static void Main(string[] argv)
    {
        OEGraphMol src = new OEGraphMol();

        OEChem.OESmilesToMol(src, "c1ccccc1");

        // make holes in the molecule index space and juggle things around
        OEChem.OEAddExplicitHydrogens(src);
        OEChem.OESuppressHydrogens(src);
        OEChem.OEAddExplicitHydrogens(src);
        OEChem.OECanonicalOrderAtoms(src);

        OEAtomBasePtrArray atommap = new OEAtomBasePtrArray(src.GetMaxAtomIdx());

        OEGraphMol dst = new OEGraphMol();
        OEChem.OECopyMol(dst, src, atommap);

        foreach (OEAtomBase srcatom in src.GetAtoms())
        {
            OEAtomBase dstatom = atommap.GetItem((int)srcatom.GetIdx());
            Console.Write(srcatom.GetIdx() + " -> " + dstatom.GetIdx());
        }
    }
}
```

OECopyPDBData

```
bool OECopyPDBData(OEMolBase& dst, const OEMolBase& src)
```

Copies the entire set of PDB data from a source molecule to a destination molecule.

See also

- *PDB Tagged Data Manipulation* section

OECopySDData

```
bool OECopySDData(OEMolBase& dst, const OEMolBase& src)
bool OECopySDData(OEMolBase& dst, const OESystem::OEContainer& src)
```

Copies the entire set of SD data from a source to a destination molecule.

See also

- *SD Tagged Data Manipulation* section

OECorrectAcidProtonGeometry

```
bool OECorrectAcidProtonGeometry(OEMolBase& mol)
bool OECorrectAcidProtonGeometry(OEMolBase& mol,
                                 const OEAtomBase* atom)
```

OECount

```
unsigned int OECount(const OEMolBase& mol, const OESubSearch& ss,
                     bool uniquematch = false)
```

Returns the number of substructure search matches for the given molecule.

uniquematch
If `true` then only the unique matches will be counted. See more details in *OESubSearch.Match* method.

```
unsigned int OECount(const OEMolBase& mol,
                     const OESystem::OEUnaryPredicate<OEAtomBase> &)
```

Returns the number of atoms (*OEAtomBase*) matching the given predicate.

```
unsigned int OECount(const OEAtomBondSet& abset,
                     const OESystem::OEUnaryPredicate<OEAtomBase> &)
```

Returns the number of atoms (*OEAtomBase*) matching the given predicate from the provided container.

```
unsigned int OECount(const OEMolBase& mol,
                     const OESystem::OEUnaryPredicate<OEBondBase> &)
```

Returns the number of bonds (*OEBondBase*) matching the given predicate.

```
unsigned int OECount(const OEAtomBondSet& abset,
                     const OESystem::OEUnaryPredicate<OEBondBase> &)
```

Returns the number of bonds (*OEBondBase*) matching the given predicate from the provided container.

```
unsigned int OECount(const OEMolBase& mol,
                     const OESystem::OEUnaryPredicate<OEGroupBase> &)
```

Returns the number of groups (*OEGroupBase*) matching the given predicate.

See also

- *OECount* functions in **OESystem TK**

```
unsigned OECount(const OEMonomerSet& monomers,
                 const OESystem::OEUnaryPredicate<OEMonomer>& pred)
```

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

Returns the number of groups (*OEMonomer*) matching the given predicate.

See also

- *OEMonomerSet* class

OECreateAbsSmiString

```
void OECreateAbsSmiString(std::string& str, const OEMolBase& mol,
                          std::pair<const OEAtomBase *, const OEAtomBase *>* atmord =␣
↪(std::pair<const OEAtomBase *, const OEAtomBase *> *) 0)
```

Creates a SMILES string representing a given molecule, without isotopic labeling or stereochemistry, and in arbitrary output order (*i.e.* not necessarily canonical). This function is just a special case of the *OECreateSmiString* function.

Note

This function does not correspond to any of the forms (flavors) of SMILES string produced by *Daylight*.

See also

- *OECreateCanSmiString* to create an canonical SMILES
- *OECreateIsoSmiString* to create an canonical isomeric SMILES

OECreateCanSmiString

```
void OECreateCanSmiString(std::string& str, const OEMolBase& mol)
```

Creates a canonical SMILES string representing a given molecule, but without isotopic labeling or stereochemistry. This function is just a special case of the *OECreateSmiString* function, called with the flavor *OESMILESFlag.DEFAULT*.

Note

This function produces what Daylight Chemical Information Systems term a *unique* SMILES.

Note that the canonical SMILES generated by this function remains dependent on the state of the molecule, especially its aromaticity state (see examples in *Aromaticity Models in OEChem TK* section). Thus, to generate a canonical smiles suitable for purposes such as a database key, the user must assure that the state of the molecule has been standardized. In particular, aromaticity should be perceived according to the preferred model.

In contrast, the high-level output function *OEWriteMolecule*, when writing the canonical SMILES format (*OEFormat.CAN*) does invoke *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags*.

See also

- *OEMolToSmiles* for a high-level and **preferred** way to create SMILES equivalent to what is written by *OEWriteMolecule*.
- *OECreateAbsSmiString* to create an arbitrary SMILES
- *OECreateIsoSmiString* to create an canonical isomeric SMILES

Validation:

The *OECreateCanSmiString* is rigorously tested to ensure that it generates unique string representations of molecules. This test involves randomly reordering the atoms and bonds of molecules and confirming that this has no effect on the SMILES generated by the *OECreateCanSmiString* function.

Table 36: Performance of **OECreateCanSmiString** to generate canonical SMILES

Database	Size	Success Rate
Wombat	53K	100.0 %
Maybridge	64K	100.0 %
MDDR	111K	100.0 %
NCI	250K	100.0 %

Running the validation test on 19M unique molecules produces only in 57 failures.

OECreateCSVHeader

```
void OECreateCSVHeader(OECSVHeader& header, const OEMolBase& mol)
void OECreateCSVHeader(OECSVHeader& header, const OEMCMolBase& mol)
```

Initializes a *OECSVHeader* object to the SD data tags present in `mol`. If a *OEMCMolBase* is given, all the SD data tags across all the conformers are considered.

OECreateInChIKey

```
void OECreateInChIKey(std::string& str, const OEMolBase& mol,
                      const OEInChIOptions& = OEInChIOptions())
```

Creates an InChIKey string representing a given molecule.

Note

The underlying InChI code is v1.06 released on December 15, 2020.

OECreateInChI

```
void OECreateInChI(std::string& str, const OEMolBase& mol,
                   const OEInChIOptions& = OEInChIOptions())
```

Creates an *standard InChI* or a *nonstandard InChI* string representing a given molecule. See the *InChI validation* chapter for information on validations of generated InChI strings.

Note

Setting any options will by default return a *nonstandard InChI*.

Note

The underlying InChI code is v1.06 released on December 15, 2020.

See also

- *OEMolToSTDInChI* function
- *OEMolToInChI* function
- *OEInChIToMol* function
- *OEParseInChI* function
- *Atom Stereo From Bond Stereo* chapter for valid bond stereo configurations
- InChi Software Download

OECreateIsoSmiString

```
void OECreateIsoSmiString(std::string& str, const OEMolBase& mol)
```

Creates an canonical isomeric SMILES string representing a given molecule.

This function is just a special case of the *OECreateSmiString* function, called with the flavor *OESMILESFlag.ISOMERIC*.

Note

This function produces SMILES that correspond what *Daylight* Chemical Information Systems term an '*absolute*' SMILES

Note that the canonical SMILES generated by this function remains dependent on the state of the molecule, especially its aromaticity state (see examples in *Aromaticity Models in OEChem TK* section). Thus, to generate a canonical smiles suitable for purposes such as a database key, the user must assure that the state of the molecule has been standardized. In particular, aromaticity should be perceived according to the preferred model.

In contrast, the high-level output function *OEWriteMolecule* (and *OEMolToSmiles*) , when writing the canonical SMILES format (*OEFormat.ISM*) does invoke *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags*.

Furthermore, whether *OEWriteMolecule* or *OECreateIsoSmiString* is used, the canonical SMILES generated depends on the current stereo specifications for the molecule. If the goal is a canonical isomeric SMILES which is unique for all representations of an equivalent stereoisomer, *i.e.*, for use as a database key, it is the user's responsibility to assure that the stereochemical state of the molecule has been rationalized and standardized, using methods such as:

- *OEPerceiveChiral*, *OE3DToAtomStereo*, *OE3DToBondStereo*
- *OEAtomBase.IsChiral*, *OEAtomBase.SetStereo*
- *OEBondBase.IsChiral*, *OEBondBase.SetStereo*

See also

- *OEMolToSmiles* for a high-level way to create SMILES equivalent to what is written by *OEWriteMolecule*.
- *OECreateAbsSmiString* to create an arbitrary SMILES
- *OECreateCanSmiString* to create a canonical SMILES
- Example program *UniqMol.cs*

Validation:

The *OECreateIsoSmiString* is rigorously tested to ensure that it generates unique string representations of molecules. This test involves randomly reordering the atoms and bonds of molecules and confirming that this has no effect on the SMILES generated by the *OECreateIsoSmiString* function.

Table 37: Performance of **OECreateIsoSmiString** to generate canonical isomeric SMILES

Database	Size	Success Rate
Wombat	53K	100.0 %
Maybridge	64K	100.0 %
MDDR	111K	100.0 %
NCI	250K	100.0 %

Running the validation test on 24M unique molecules produces only 129 failure cases.

OECreateMolDatabaseIdx

```
bool OECreateMolDatabaseIdx(const std::string& databaseFileName)
bool OECreateMolDatabaseIdx(const std::string& databaseFileName,
                            const std::string& indexFileName)
```

Creates an index file parallel to the file named by `databaseFileName` to improve the performance of *OEMolDatabase.Open* as described by the *Index Files* section. If *OEMolDatabase.Open* detects the presence of a parallel `.idx` file, the file position offsets will be read from the `.idx` file instead of having to stream the entire file off of disk to find the offsets during the call to *OEMolDatabase.Open*.

The index file name created is based upon the `databaseFileName` passed, optionally accounting for `.gz`. The following table demonstrates the pattern used to create index file names:

Database File Name	Index File Name
database.oeb	database.oeb.idx
database.oeb.gz	database.oeb.idx.gz

The overload of the function that takes an `indexFileName` will write the index to that file name instead.

Returns `false` if the function fails to create the index file for any reason, e.g., unable to open the file specified by `databaseFileName`.

Example

Listing 1: Creates an index file

```
        String ifname = "drugs.sdf";
        String idxfname = OEChem.OEGetMolDatabaseIdxFileName(ifname);

        if (!File.Exists(idxfname)) {
            if (!OEChem.OECreateMolDatabaseIdx(ifname))
                OEChem.OEThrow.Warning("Unable to create " + idxfname + "  molecule␣
↪index file");
        }
        OEChem.OEThrow.Info("Using " + idxfname + " index molecule file");
```

Note

`.idx` files can be concatenated together. Care must be taken to concatenate the corresponding database files together in the exact same order.

OECreateSlnString

```
void OECreateSlnString(std::string& str, const OEMolBase& mol,
                       unsigned int flavor = OESLNFlag::DEFAULT)
```

Creates a *Tripos* SLN string representing a given molecule. Several variants of SLN format are supported by using different flavors from the *OESLNFlag* namespace.

OECreateSmiString

```
void OECreateSmiString(std::string& str, const OEMolBase& mol,
                       unsigned int flavor = OESMILESFlag::DEFAULT,
                       std::pair<const OEAtomBase*, const OEAtomBase*>* atmord =␣
↪(std::pair<const OEAtomBase*, const OEAtomBase*>*)0)
```

Creates a SMILES string representing a given molecule. Several variants of SMILES format are supported by using different flavors from the *OESMILESFlag* namespace.

Note

OEGetSmiStringOrder function can be used to return an iterator over atoms, in the order in which they appear in the SMILES string.

Table 38: **The following function calls generate identical SMILES**

OEChem TK function	calling *OECreateSmiString* with flags from the *OESMILESFlag* namespace
OECreateAbsSmiStr	*AtomMaps* | *RGroups*
OECreateCanSmiStr	*DEFAULT* identical to *Canonical* | *AtomMaps* | *RGroups*
OECreateIsoSmiStr	*ISOMERIC* identical to *Isotopes* | *AtomStereo* | *BondStereo* | *Canonical* | *AtomMaps* | *RGroups*
OEMolToSmiles	*ISOMERIC* identical to *Isotopes* | *AtomStereo* | *BondStereo* | *Canonical* | *AtomMaps* | *RGroups*

[1] *OEMolToSmiles* is a high-level function that perceives necessary properties before generating canonical isomeric SMILES

OECreateSubSearchDatabaseFile

> **Attention**
>
> This API is currently available in C++ and Python.

```
bool OECreateSubSearchDatabaseFile(const std::string& ofname,
                                   const std::string& ifname,
                                   const OESubSearchScreenTypeBase* stype,
                                   OESystem::OETracerBase& tracer = OESystem::OENoTracer)
bool OECreateSubSearchDatabaseFile(const std::string& ofname,
                                   const std::string& ifname,
                                   const OECreateSubSearchDatabaseOptions& opts,
                                   OESystem::OETracerBase& tracer = OESystem::OENoTracer)
```

Generates substructure search database files for *OESubSearchDatabase*. *OECreateSubSearchDatabaseFile* and returns `true` if the database is successfully created.

ofname
: The name of the generated substructure search database file. The generated file will be an OpenEye binary file (*OEFormat.OEB*); therefore, `.oeb` file extension is required in the given filename.

ifname
: The name of the input molecule file.

stype
: The type of the screen that will be pre-generated to each molecule.

opts
: The *OECreateSubSearchDatabaseOptions* object that encapsulates properties that determine how the database is generated.

tracer
: A tracer that can be used to report the progress of the database generation.

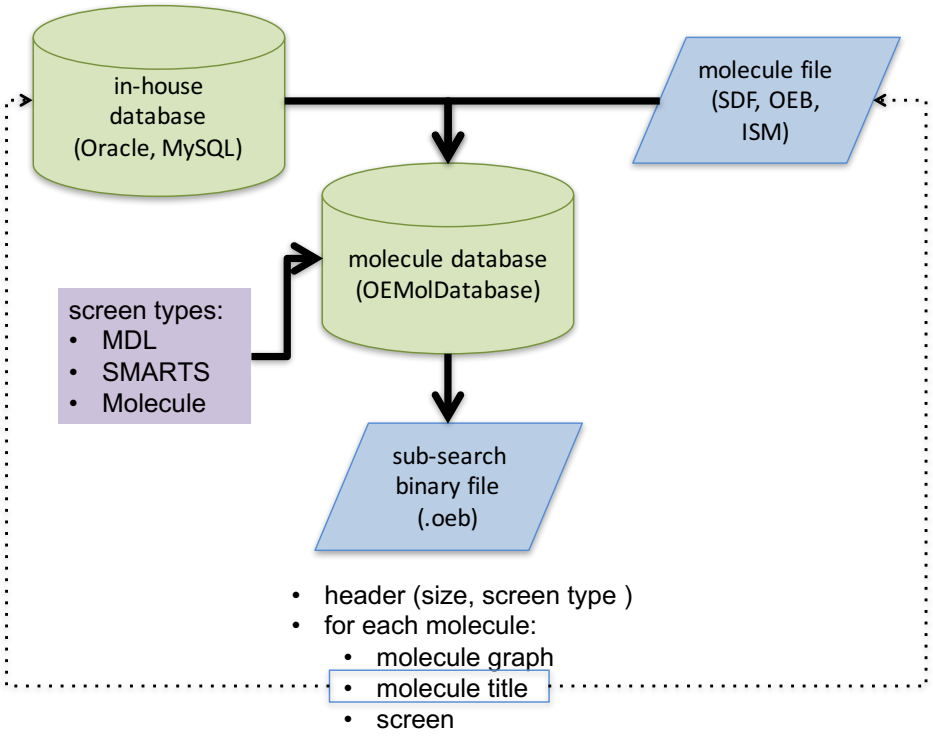


Fig. 9: **Schematic representation of substructure search database file generation process**

The generated database file is an OpenEye binary file with a header that stores extra information about the database such as, screen type. Each substructure database file is associated with one and only one screen type. The molecules will be pre-processed for that specific type of substructure search. In order to minimize the memory footprints of the dataset at search time, all unessential data will be stripped from the molecules and explicit hydrogens will be suppressed. The search database will store only the following information for each molecule:

- the molecular graph (no coordinates)
- the molecule title (either user-defined or a unique identifier assigned during database generation)
- the screen generated for the molecule

The titles are kept in order to provide an external id for each molecule. See also `OECreateSubSearchDatabaseOptions.SetKeepTitle`. When searching the database, the titles of the matched molecules can be retrieved by the `OESubSearchDatabase.GetMatchTitles` method.

Note

If a sorted database is generated (see the `OECreateSubSearchDatabaseOptions.SetSortByBitCounts` method), a temporary file will be written into the disk prior to generating the final sorted database. This two-step process allows to generate database files that can not be fit into the memory with the size that is limited by the disk space.

See also

- *OECreateSubSearchDatabaseOptions* class

- *OESubSearchScreenTypeBase* class
- *OEConsoleProgressTracer* class

Code Example

- *Generate database for fast substructure search* example

OEDefaultImplicitHCount

```
unsigned int OEDefaultImplicitHCount(const OEAtomBase* atm)
```

Determines the default number of implicit hydrogens on an atom using **OpenEye's valence model**. This count assumes that it is free to change the formal charge of an atom.

Note

If the charge state is correctly set on each atom, you should call the *OEDefaultMDLHCount* function instead.

See also

- *OpenEye Valence Model* section
- *OEAssignImplicitHydrogens* function

OEDefaultMDLHCount

```
unsigned int OEDefaultMDLHCount(const OEAtomBase* atm)
```

Determines the default number of implicit hydrogens on an atom using **MDL's valence model**. This count assumes that the formal charge on an atom has been correctly set. If the formal charge on each atom is not know *a priori*, the *OEDefaultImplicitHCount* function may be more suitable.

See also

- *MDL Valence Model* section about MDL valence model
- *OEAssignMDLHydrogens* function
- *OEMDLGetValence* function
- *OEIsValidMDLAtomValence* predicate

OEDeleteEverythingExceptTheFirstLargestComponent

```
bool OEDeleteEverythingExceptTheFirstLargestComponent(OEMolBase & mol)
```

Deletes all atoms that are not part of the largest connected component of a molecule. If two or more components of a molecule are the size of the largest component, this function keeps the first one encoun-

tered. Only heavy atoms are taken into consideration when determining the size of the components. The *OEDeleteEverythingExceptTheFirstLargestComponent* function returns `false` if the given molecule has only one connected component, otherwise it returns `true`.

Note

The *OEDeleteEverythingExceptTheFirstLargestComponent* function is identical to *OETheFunctionFormerlyKnownAsStripSalts*.

See also

- Example program *StripSalts.cs*

OEDeletePDBData

```
bool OEDeletePDBData(OEMolBase& mol, const std::string& tag)
```

Delete all PDB data items with the specified tag.

See also

- *OEClearPDBData* function to remove all PDB data
- *PDB Tagged Data Manipulation* section

OEDeleteSDData

```
bool OEDeleteSDData(OEMolBase& mol, const std::string& tag)
```

Delete all SD data items with the specified tag.

See also

- *OEClearSDData* to remove all SD data.
- *SD Tagged Data Manipulation* section

Code Example

- *Modifying SD tags*

OEDetermineAromaticRingSystems

```
unsigned int OEDetermineAromaticRingSystems(const OEMolBase& mol,
                                            unsigned int* rings)
```

Determines the aromatic ring systems of a molecule. The return value is the number of aromatic ring systems found in the molecule. If the molecule is acyclic or there is no aromatic ring system in the molecule, the return value is

zero. The 'rings' array must contain at least *OEMolBase.GetMaxAtomIdx* elements, and upon return it contains a mapping indexed by *OEAtomBase.GetIdx* of which aromatic ring system each atom is in. The aromatic ring systems are numbered from one, up to an including the return value of the function. Atoms that aren't contained in an aromatic ring, *i.e.* for which *OEAtomBase.IsInRing* or *OEAtomBase.IsAromatic* returns `false`, are mapped to the value zero.

Warning

It is necessary to call *OEFindRingAtomsAndBonds* and *OEAssignAromaticFlags* functions prior to calling *OEDetermineAromaticRingSystems* function.

See also

- *Ring Systems Identification* section

OEDetermineComponents

```
unsigned int OEDetermineComponents(const OEMolBase& mol,
                                   unsigned int* parts)
```

Determines the connected components of a molecule. The return value is the number of connected components found in the molecule. If the molecule contains a single discrete molecule, the return value is one. The 'parts' array must contain at least *OEMolBase.GetMaxAtomIdx* elements, and upon return it contains a mapping indexed by *OEAtomBase.GetIdx* of which part or connected component each atom is in. The parts are numbered from one, up to and including the return value of the function.

See also

- *Connected Components Identification* section
- Example program *Parts2Mols.cs*

OEDetermineConnectivity

```
void OEDetermineConnectivity(OEMolBase& mol, bool same_altloc = true)
```

Perceives the covalent bonds of a molecule from 3D coordinates. This routine creates a bond between any pair of atoms that are closer than the sum of their covalent radii plus a slop factor of `0.45` Angstroms. The covalent radii of each atom are as defined by the Cambridge Crystallographic Database (http://www.ccdc.cam.ac.uk), and are those returned by the **OEChem TK** function *OEGetCovalentRadius*. Bonds are not created by atoms separated by less than `0.4` Å. All bonds are created with bond order 1. The **OEChem TK** function *OEPerceiveBondOrders* may be used to assign bond orders from the 3D geometry and connectivity.

See also

- *Connectivity Perception* chapter and *Bond Order Perception* section

OEDetermineReactants

```
unsigned int OEDetermineReactants(const OEMolBase& mol,
                                  unsigned int* parts)
```

OEDetermineRingSystems

```
unsigned int OEDetermineRingSystems(const OEMolBase& mol,
                                    unsigned int* rings)
```

Determines the ring systems of a molecule. The return value is the number of ring systems found in the molecule. If the molecule is acyclic, the return value is zero. The 'rings' array must contain at least *OEMolBase.GetMaxAtomIdx* elements, and upon return it contains a mapping indexed by *OEAtomBase.GetIdx* of which ring system each atom is in. The ring systems are numbered from one, up to an including the return value of the function. Atoms that aren't contained in a ring, *i.e.* for which *OEAtomBase.IsInRing* returns `false`, are mapped to the value zero.

Warning

It is necessary to perform ring perception by calling *OEFindRingAtomsAndBonds* function prior to calling *OEDetermineRingSystems* function.

See also

- *Ring Systems Identification* section

OEDisassembleExpressions

```
bool OEDisassembleExpressions(OEQMolBase &)
```

Warning

This function is deprecated. The activities performed by this function will be internalized in a future release.

OEDoubleBondCount

```
unsigned int OEDoubleBondCount(const OEAtomBase* atm)
```

Returns the number of double bonds connected to an atom.

OEDoubleBondOCount

```
unsigned int OEDoubleBondOCount(const OEAtomBase* atm)
```

OEElementHistogram

OEEulerRotate

```
void OEEulerRotate(OEMolBase& mol, const double* angles)
void OEEulerRotate(OEConfBase& conf, const double* angles)
void OEEulerRotate(OEMCMolBase& mcmol, const double* angles)
```

Rotates the molecules about the z, x', and z' axes in order. It is assumed that the angles array is at least of length 3. The angles array should contain the z rotation in radians in `angles[0]`, the x' rotation in radians in `angles[1]`, and the z' rotation in radians in `angles[2]`. The overloads for *OEConfBase* and *OEMCMolBase* are for efficiency.

mol
: The molecule to rotate.

conf
: The conformer to rotate.

mcmol
: The multi-conformer molecule to rotate.

angle
: The rotation angle.

OEExactGraphMatch

```
bool OEExactGraphMatch(const OEMolBase& mol1, const OEMolBase& mol2)
```

Determines whether two molecules, 'mol1' and 'mol2', have exactly the same graph. This function effectively compares the canonical SMILES of both molecules (created by *OECreateCanSmiString*) using string equality.

OEExpandSuperAtoms

```
bool OEExpandSuperAtoms(OEMolBase& mol)
```

Expands (or instantiates) the recognized super-atoms in a connection table. Superatoms are commonly found in MDL mol files and some SMILES variants as a way of specifying a common functional group by a single superatom. This function replaces each pseudo atom with the set of atoms and bonds it represents.

Currently this function expands the 20 naturally occurring amino acids and 'Abu' (2-amino butyric acid). Additional superatoms, such as common protecting groups, may be supported in the future. Unrecognized superatoms are unmodified by this function.

This function determines connectivity for multi-valent superatoms from 2D coordinates when available, and failing that from atom indices, *i.e.* the order in which atoms are created. These heuristics allow correct interpretation of most common uses of superatoms, such as the SMILES variant `[Ala][Asn][Thr]`.

OEExtractRingTemplates

```
OESystem::OEIterBase<OEMolBase> * OEExtractRingTemplates(const OEMolBase& mol)
OESystem::OEIterBase<OEMolBase> * OEExtractRingTemplates(const OEMolBase& mol,
                                                         const␣
↪OEExtractRingTemplateOptions& opts)
```

Returns an iterator over the anonymized ring systems of the given molecule.

Note

The *OEExtractRingTemplates* function is used to extract potential new ring templates for the 2D coordinate generation system and it requires an input molecule with 2D coordinates.

We recommend using the *OE2DRingDictionary* class for generating a ring template dictionary.

See also

- *OEExtractRingTemplateOptions* class
- Example program *ExtractRingTemplates.cs*

OEFastaToMol

```
bool OEFastaToMol(OEMolBase& mol, const std::string& str)
```

Converts a FASTA sequence into a molecule.

The following code demonstrates how to create a molecule from a FASTA sequence using the *OEFastaToMol* function and prints out the number of alpha carbon atoms in the molecule.

```
        OEGraphMol mol = new OEGraphMol();

        OEChem.OEFastaToMol(mol, "AVILMPTWSTNQCGPRHKDE");
        Console.WriteLine(OEChem.OECount(mol, new OEIsCAlpha()));
```

See also

- *OEReadFASTAFile* function

OEFindRingAtomsAndBonds

```
void OEFindRingAtomsAndBonds(OEMolBase& mol)
```

Determines which atoms and bonds of a molecule are contained in rings, and set their 'in ring' property appropriately via *OEAtomBase.SetInRing* and *OEBondBase.SetInRing* methods, respectively.

See also

- *Ring Perception* chapter
- *Connectivity Perception* chapter and *Bond Order Perception* section
- Example program *CanSmi.cs*

OEFormalPartialCharges

```
void OEFormalPartialCharges(OEMolBase& mol)
```

Set the partial charge property of each atom in a molecule to its formal charge. This is equivalent of calling the *OEAtomBase.SetPartialCharge* method on every atom with the value returned from *OEAtomBase.GetFormalCharge*.

OEGasteigerInitialCharges

```
bool OEGasteigerInitialCharges(OEMolBase& mol)
```

Calculates the seed charge partial charges used by the Marsilli-Gasteiger partial charge algorithm.

Gasteiger initial charges assign the partial charge of `-0.5` to atoms of type *OETriposType.Oco2*, *i.e.* carboxylate oxygens, and use the formal charge as the partial charge on all other atoms except those in conjugated rings. For conjugated ring systems, consisting of Tripos atom types *OETriposType.C2*, *OETriposType.Car*, *OETriposType.N2*, *OETriposType.Nar*, *OETriposType.Npl3* and *OETriposType.Nam*, the sum of the formal charges of the atoms in the ring system are divided equally amongst all atoms in the ring system. For example, the five heavy atoms in imidazolium (`[nH+]1c[nH]cc1`), each get a partial charge of `0.2`. O

Warning

It is not necessary to call *OEGasteigerInitialCharges* function prior to calling *OEGasteigerPartialCharges* function.

See also

- *Gasteiger Partial Charge Calculation* section
- *OETriposAtomTypes* to determine Tripos atom types

OEGasteigerPartialCharges

```
bool OEGasteigerPartialCharges(OEMolBase& mol)
```

Calculate Marsilli-Gasteiger partial charges for a molecule. The results of the calculation are placed in the partial charge property of each atom using *OEAtomBase.SetPartialCharge*.

See also

- *OEGasteigerInitialCharges*
- *Gasteiger Partial Charge Calculation* section

OEGenerate2DCoordinates

```
bool OEGenerate2DCoordinates(OEMolBase& mol)
```

Assigns 2D coordinates to the given *OEMolBase* object. The coordinates of each explicit atom are assigned using the *OEMolBase.SetCoords* method. This function automatically set the dimensionality of the molecule to two, using the

OEMolBase.SetDimension method.

Note

OEGenerate2DCoordinates is a low-level function that only generates 2D coordinates but does not perform any other perceptions.

OEGenerate2DCoordinates will clear MDL bond stereo information that can be re-perceived by calling the *OEMDLPerceiveBondStereo* function.

If *OEGenerate2DCoordinates* fails to generate 2D coordinates that correspond to the atom and bond stereo information of the molecule it returns `false`. This problem occurs when no **built-in** ring template exists for a specific stereo configuration. To avoid this issue specifying user-defined ring template might be necessary. For more information see the *2D Coordinate Generation* chapter.

See also

OEInit2DRingDictionary function

OEGetAbsTorsion

```
double OEGetAbsTorsion(const OEMolBase& mol, const OEAtomBase* a,
                       const OEAtomBase* b, const OEAtomBase* c,
                       const OEAtomBase* d)
double OEGetAbsTorsion(const OEConfBase& conf, const OEAtomBase* a,
                       const OEAtomBase* b, const OEAtomBase* c,
                       const OEAtomBase* d)
```

Returns the absolute value of the torsion defined by atoms *a*, *b*, *c*, and *d*. The return value is in radians and varies from `0` to +`PI`. This function returns the same value as fabs(*OEGetTorsion*), yet it is more efficient. These functions assume that the four atoms are members of the molecule or conformer which is passed to the function. The *OEConfBase* overload is strictly for efficiency.

mol
: The molecule that the pair of atom belongs to.

conf
: The conformer that the pair of atom belongs to.

a
: One of the end atoms in the torsion.

b
: Atom that is connected to *a* in the torsion.

c
: Atom that is connected to *d* in the torsion.

d
: The other end atom in the torsion.

OEGetAminoAcidCode

```
char OEGetAminoAcidCode(unsigned int idx)
```

Returns the single character amino acid code corresponding to the specified residue (using the *OEResidueIndex* namespace format). The character `X` is returned for the residue *OEResidueIndex.UNK* and the NULL character (0) is returned for all other non-amino acid residues.

See also

- *OEGetResidueIndexFromCode* inverse function
- *OEResidueIndex* namespace

OEGetAngle

```
double OEGetAngle(const OEMolBase& mol, const OEAtomBase* a,
                  const OEAtomBase* b, const OEAtomBase* c)
double OEGetAngle(const OEConfBase& conf, const OEAtomBase* a,
                  const OEAtomBase* b, const OEAtomBase* c)
double OEGetAngle(const OEMolBase& ma, const OEAtomBase* a,
                  const OEMolBase& mb, const OEAtomBase* b,
                  const OEMolBase& mc, const OEAtomBase* c)
double OEGetAngle(const OEConfBase& ca, const OEAtomBase* a,
                  const OEConfBase& cb, const OEAtomBase* b,
                  const OEConfBase& cc, const OEAtomBase* c)
```

Returns the angle formed by three atoms where the atom passed in as the *b* argument is the vertex. The function which takes a single *OEMolBase* argument assumes that all three atoms are contained in the single molecule. The *OEConfBase* overload functions are not strictly necessary, but are useful for efficiency.

mol
: The molecule that the atoms belong to (when all atoms belong to the same molecule).

conf
: The conformer that the atoms belong to (when all atoms belong to the same conformer).

a
: One of the end atoms.

b
: The vertex atom.

c
: One of the end atoms.

ma
: The molecule that atom *a* belongs to (when the atoms are in three different molecule).

mb
: The molecule that atom *b* belongs to (when the atoms are in three different molecule).

mc
: The molecule that atom *c* belongs to (when the atoms are in three different molecule).

ca
: The conformer that atom *a* belongs to (when the atoms are in three different conformer).

cb
The conformer that atom *b* belongs to (when the atoms are in three different conformer).

cc
The conformer that atom *c* belongs to (when the atoms are in three different conformer).

OEGetAnisou

```
bool OEGetAnisou(OEAnisoUij& uij, const OEAtomBase* atom)
bool OEGetAnisou(const OEAtomBase* atom, int& u11, int& u22, int& u33,
                 int& u12, int& u13, int& u23)
```

This function is used to return anisotropic B-factor parameters associated with an *OEAtomBase* from a PDB file read with the flavor *OEIFlavor.PDB.DATA*.

If the atom does not have associated anisotropic B-factor parameters, returns `false` and sets all the parameter values to 0. The predicate *OEHasAnisou* can be used to check for parameters.

The form that uses a *OEAnisoUij* returns the data in packaged form for convenience in wrapped languages.

See also

- *OEAnisoUij*
- *OEHasAnisou*
- *OESetAnisou*

OEGetAtomColor

```
OESystem::OEColor OEGetAtomColor(const OEChem::OEAtomBase& atom)
```

Returns the OEColor stored in the *OE3DMolStyle* of the molecule asscoiated with the atom specified.

OEGetAtomComment

```
const char* OEGetAtomComment(const OEAtomBase* atm)
```

Used to retrieve the text comment associated with an individual atom. This field is used to manipulate the atom alias information in MDL file formats. This property may be set using the *OESetAtomComment* function.

OEGetAtomLabelColor

```
OESystem::OEColor OEGetAtomLabelColor(const OEChem::OEAtomBase& atom)
```

Returns the OEColor stored in the *OE3DMolStyle* for the atom label of the atom specified.

OEGetAtomLabelText

```
std::string OEGetAtomLabelText(const OEChem::OEAtomBase& atom)
```

Returns a string of the label associated with specified atom.

OEGetAtomicNum

```
unsigned int OEGetAtomicNum(const char* sym)
```

Returns the atomic number associated with the specified atomic symbol. The specified buffer `sym` must contain a string terminated by either a space or a NULL, `\0`, character. Zero-length strings, strings longer than two characters or unrecognized symbols all make the function return `0`. The comparison is case insensitive, such that the first character need not be uppercase, and the second character (if one exists) need not be lowercase.

This function returns the value 1 for symbols 'H', 'D' and 'T'.

OEGetAtomicSymbol

```
const char* OEGetAtomicSymbol(unsigned int elemno)
```

Returns the atomic symbol associated with the specified atomic number. This function returns a valid pointer to a NUL-character, `\0`, if `elemno` is neither greater than zero nor less than the symbolic constant *OEElemNo.MAXELEM*. Otherwise, all returned values are NUL-terminated, with the first character uppercase, and the second character either a NUL or lowercase (in which case the third character is always the NUL-terminator).

OEGetAutomorphs

```
OESystem::OEIterBase<OEMatchBase>* OEGetAutomorphs(const OEMolBase& mol,
                                                   bool includeH = false)
```

Returns an iterator over all the automorphisms, i.e. self matches of the molecule.

includeH
: This flag indicates whether explicit hydrogen atoms should be considered when assessing automorphisms. Default is `false`.

OEGetAverageWeight

```
double OEGetAverageWeight(unsigned int elemno)
```

Returns the average atomic weight of an element, based upon the naturally occurring abundances of its isotopes. To return the 'monoisotopic' weight for a particular element use the *OEGetIsotopicWeight* function like the following.

```
        uint elemno = OEElemNo.O;
        double weight = OEChem.OEGetIsotopicWeight(elemno, OEChem.
↪OEGetDefaultMass(elemno));
```

OEGetBondiVdWRadius

```
double OEGetBondiVdWRadius(unsigned int elemno)
```

Returns the Van der Waals radius for a particular atomic number as tabulated by Bondi [Bondi-1964]. There are several experimental values for Van der Waals radii (see also *OEGetPaulingVdWRadius*) and many additional: values used in molecular mechanics forcefields (including united-atom VdW radii), so there is often no definitive notion of Van der Waals radius.

OEGetCenterAndExtents

```
void OEGetCenterAndExtents(const OEMolBase& mol, float* ctr,
                           float* ext)
Determines the bounding box that can encompass the molecule. The length of
each side of the box is output in the 3-element array extents. The geometric
center of the box is output to the center parameter.
*mol*The molecule to calculate center and extents.
*ctr*The output box center vector.  The vector must have a defined size of at least␣
↪three.
*ext*The output box lengths vector.  The vector must have a defined size of at least␣
↪three.
```

Note

The center of the box is different from the center-of-mass of the molecule.

OEGetCenterOfMass

```
void OEGetCenterOfMass(const OEMolBase& mol, float* com,
                       bool weightAtoms = false)
```

Calculates the center-of-mass of the molecule. The `weightAtoms` can be specified to weight each of the atom coordinates with the corresponding atom weights.

mol

The molecule to calculate center-of-mass.

com

The output center-of-mass array. The array must have a defined size of at least three.

weightAtoms

Flag indicating if the atom weights should be used to weight the coordinates during calculation.

OEGetColorSchemeName

```
const char* OEGetColorSchemeName(unsigned int colorScheme)
```

Returns the color scheme name as a const char. The function is called in the *OEMolStyleColorer.GetPropertyName* function of *OEMolStyleColorer*. It can be used in tandem with the *OE3DMolStyle.GetAtomColorer* function of *OE3DMolStyle*.

example: std::string str = style.GetAtomColorer().GetPropertyName();

OEGetComment

```
const char* OEGetComment(const OEMolBase& mol)
```

Returns a pointer to the text comment attached to a molecule. These comments are written to the comment field of .sdf and .mol2 formats and are also stored in OEBinary format. This property may be set using the *OESetComment* function.

OEGetCovalentRadius

```
double OEGetCovalentRadius(unsigned int elemno)
```

Returns the covalent radius of the element specified by the given atomic number. The covalent radius of an atom is typically used in bonding calculations, such as the function *OEDetermineConnectivity*. A value of zero is returned for elements other than those commonly found in organic molecules. The actual values are those prescribed by the Cambridge Crystallographic Database (http://www.ccdc.cam.ac.uk).

OEGetDefaultIFlavor

```
unsigned int OEGetDefaultIFlavor(unsigned int format)
```

Returns the default input file flavor associated with the given file format. The set of valid file formats are defined in the *OEFormat* namespace. The default flavors are automatically set by the *oemolistream* constructors. The flavor for any format can be set using the *oemolstreambase.SetFlavor* method.

See also

- *OESetDefaultFlavors* function.
- *OESetDefaultIFlavors* function.
- *OEIFlavor* and *the OEFormat namespace*.
- *oemolstreambase.SetFlavor* and *oemolstreambase.GetFlavor* methods.

OEGetDefaultOFlavor

```
unsigned int OEGetDefaultOFlavor(unsigned int format)
```

Returns the default output file flavor associated with the given file format. The set of valid file formats are defined in the *OEFormat* namespace. The default flavors are automatically set by the *oemolostream* constructors. The flavor for any format can be set using the *oemolstreambase.SetFlavor* method.

See also

- *OESetDefaultFlavors* function
- *OESetDefaultOFlavors* function
- *OEOFlavor* and *the OEFormat namespace*
- *oemolstreambase.SetFlavor* and *oemolstreambase.GetFlavor* methods

OEGetDefaultMass

```
unsigned int OEGetDefaultMass(unsigned int elemno)
```

Returns the most abundant isotope for the given atomic number *i.e.* the same values as required by the MDL file format. In MDL SD files, non-natural isotopes are represented as a delta from the most commonly occurring isotope for a particular element/atomic number.

For example, for *OEElemNo.C* this function returns the value 12. For input value of zero and for values greater than or equal to *OEElemNo.MAXELEM*, it returns zero.

OEGetDelphiRadius

```
double OEGetDelphiRadius(unsigned int elemno)
```

Returns the default radius for a given element used by the Accelrys' Delphi program for electrostatics calculations.

OEGetDimensionFromCoords

```
unsigned int OEGetDimensionFromCoords(const OEMolBase& mol)
```

Returns the molecule's dimension from the coordinates. The returned value is 0, 1, 2 or 3 depending upon the number of coordinates for which any atom has a non-zero value.

The table below illustrates the difference between the *OEGetDimensionFromCoords* and *OEGetMDLDimensionFromCoords* functions.

Coordinates			OEGetDimensionFromCoords	OEGetMDLDimensionFromCoords
X	Y	Z		
0.00	0.00	0.00	0D	0D
1.00	0.00	0.00	1D	2D
0.00	1.00	0.00	1D	2D
0.00	0.00	1.00	1D	3D
1.00	1.00	0.00	2D	2D
0.00	1.00	1.00	2D	3D
1.00	0.00	1.00	2D	3D
1.00	1.00	1.00	3D	3D

See also

- *OESetDimensionFromCoords* function
- *OEGetMDLDimensionFromCoords* and *OESetMDLDimensionFromCoords* functions

OEGetDistance2

```
double OEGetDistance2(const OEMolBase& mol, const OEAtomBase* a,
                      const OEAtomBase* b)
double OEGetDistance2(const OEConfBase& conf, const OEAtomBase* a,
                      const OEAtomBase* b)
double OEGetDistance2(const OEMolBase& ma, const OEAtomBase* a,
                      const OEMolBase& mb, const OEAtomBase* b)
double OEGetDistance2(const OEConfBase& ca, const OEAtomBase* a,
                      const OEConfBase& cb, const OEAtomBase* b)
```

These four functions are almost identical to the *OEGetDistance* function. However, they return the square of the distance between two atoms rather than distance. When it is possible to use the square of the distance, this is more efficient.

OEGetDistance

```
double OEGetDistance(const OEMolBase& mol, const OEAtomBase* a,
                     const OEAtomBase* b)
double OEGetDistance(const OEConfBase& conf, const OEAtomBase* a,
                     const OEAtomBase* b)
double OEGetDistance(const OEMolBase& ma, const OEAtomBase* a,
                     const OEMolBase& mb, const OEAtomBase* b)
double OEGetDistance(const OEConfBase& ca, const OEAtomBase* a,
                     const OEConfBase& cb, const OEAtomBase* b)
```

Calculates the distance between two atoms. The first two functions which take a single molecule (either *OEMolBase* or *OEConfBase*) assume that the two atoms are both in the single molecule. If the atoms are in different molecules, the function which takes two molecule arguments should be used. *OEConfBase* inherits from *OEMolBase* and strictly speaking the overload is not necessary. However, the specific *OEConfBase* implementation can be more efficient.

mol
The molecule that the pair of atom belongs to (when all atoms belong to the same molecule).

conf
The conformer that the pair of atom belongs to (when all atoms belong to the same conformer).

a
The first atom.

b
The second atom.

ma
The molecule that the first atom (atom *a*) belongs to (when the atoms are in two different molecules).

mb
The molecule that the second atom (atom *b*) belongs to (when the atoms are in two different molecules).

ca
The conformer that the first atom (atom *a*) belongs to (when the atoms are in two different conformers).

cb
The conformer that the second atom (atom *b*) belongs to (when the atoms are in two different conformers).

OEGetDoubleBondNeighbor

```
OEAtomBase* OEGetDoubleBondNeighbor(const OEAtomBase* atm)
```

OEGetFileExtension

```
const char* OEGetFileExtension(const char* fname)
```

Returns the file extension for the file name ‘fname’.

OEGetFileType

```
unsigned int OEGetFileType(const char* ext)
```

Returns a constant from the *the OEFormat namespace* for the file extension ‘ext’.

OEGetFormatExtension

```
const char* OEGetFormatExtension(unsigned int tag)
```

Returns a comma-separated list of possible file extensions corresponding to the specified parameter. The parameter, 'tag', should be drawn from the *the OEFormat namespace*.

See also

- Example program *OEChemInfo.cs*

OEGetFormatString

```
const char* OEGetFormatString(unsigned int tag)
```

Returns the name of the file format associated with the symbolic constant 'tag' from the *the OEFormat namespace*.

See also

- Example program *OEChemInfo.cs*

OEGetHonigIonicCavityRadius

```
double OEGetHonigIonicCavityRadius(unsigned int elemno)
```

Returns the effective ionic radius (defined in [Rashin-1985]) of each element to be used in solvation calculations.

OEGetHybridization

```
unsigned int OEGetHybridization(const OEAtomBase *)
```

This function predicts the atomic hybridization/geometry of an atom given its immediate connectivity. The return value is taken from the namespace *OEHybridization*.

Warning

The aromaticity has to be perceived by calling *OEAssignAromaticFlags* function prior to calling *OEGetHybridization* function.

OEGetHydrogenCounts

```
bool OEGetHydrogenCounts(OEHydrogenCounter& counter,
                         const OEMolBase& mol)
```

Populates an *OEHydrogenCounter* object containing the explicit and implicit count of hydrogens in the molecule. Returns true if molecule is valid and counter is populated, false otherwise.

See also

- *OEHydrogenCounter*

OEGetIsotopicWeight

```
double OEGetIsotopicWeight(unsigned int elemno, unsigned int mass)
```

Returns the atomic weight for a specific isotope. The isotope is specified by the atomic number ('elemno') and the atomic mass ('mass').

To get the average atomic weight of an element, based upon typically occurring abundances of its natural isotopes, use the *OEGetAverageWeight* function.

See also

- *OEGetAverageWeight* function

OEGetMCSExhaustiveSearchTruncationLimit

```
unsigned int OEGetMCSExhaustiveSearchTruncationLimit()
```

Returns the number of unique partial substructures that can be traversed during the *exhaustive* MCS search.

See also

- *OESetMCSExhaustiveSearchTruncationLimit* function

OEGetMDLDimensionFromCoords

```
unsigned int OEGetMDLDimensionFromCoords(const OEMolBase& mol)
```

Returns the molecule's dimension from the coordinates, The returned value is:

- `0` if all x, y, z coordinates are zero
- `2` if there is at least one atom with non-zero x or z coordinates
- `3` it there is at least one atom with non-zero z coordinate

The table below illustrates the difference between the *OEGetDimensionFromCoords* and *OEGetMDLDimensionFromCoords* functions.

Coordinates X	Y	Z	OEGetDimensionFromCoords	OEGetMDLDimensionFromCoords
0.00	0.00	0.00	0D	0D
1.00	0.00	0.00	1D	2D
0.00	1.00	0.00	1D	2D
0.00	0.00	1.00	1D	3D
1.00	1.00	0.00	2D	2D
0.00	1.00	1.00	2D	3D
1.00	0.00	1.00	2D	3D
1.00	1.00	1.00	3D	3D

See also

- *OESetMDLDimensionFromCoords* function
- *OEGetDimensionFromCoords* and *OESetDimensionFromCoords* functions

OEGetMolDatabaseIdxFileName

```
std::string OEGetMolDatabaseIdxFileName(const std::string& filename)
```

Returns the default index file name generated by *OECreateMolDatabaseIdx* and *OEMolDatabase.Save*. Returns an empty string if an empty string is given.

The index file name created is based upon the `filename` passed, optionally accounting for `.gz`. The following table demonstrates the pattern used to create index file names:

Database File Name	Index File Name
database.oeb	database.oeb.idx
database.oeb.gz	database.oeb.idx.gz

See also

- *OECreateMolDatabaseIdx*
- *OEMolDatabase.Save*

OEGetMolDBIdx

```
unsigned int OEGetMolDBIdx(const OESystem::OEBase& mol)
```

Returns the index used to retrieve the molecule from a *OEMolDatabase*. Returns `(unsigned int)-1` if `mol` did not actually originate from a *OEMolDatabase*.

OEGetNearestNbrs

```
OESystem::OEIterBase<OENbrs>* OEGetNearestNbrs(const OEMolBase& mol,
                                               double maxdist,
                                               unsigned int method =␣
↪OENearestNbrsMethod::Default)
```

Returns an iterator over all atom pairs of the given molecule that are closer to each other than the given limit.

maxdist
The maximum distance between two atoms.

method
Determines the algorithm that is used to find the nearest neighbors. This value has to be from the *OENearestNbrsMethod* namespace.

```
OESystem::OEIterBase<OENbrs>* OEGetNearestNbrs(const OEMolBase& mol,
                                               const OEAtomBase* atom,
                                               double maxdist,
                                               unsigned int method =␣
↪OENearestNbrsMethod::Default)
```

Returns an iterator over all neighbors of the given atom that is closer than the given limit.

atom
This atom does not have to belong to the given molecule.

maxdist
The maximum distance between two atoms.

method
Determines the algorithm that is used to find the nearest neighbors. This value has to be from the *OENearestNbrsMethod* namespace.

```
OESystem::OEIterBase<OENbrs>* OEGetNearestNbrs(const OEMolBase& mol,
                                               const float* coords,
                                               double maxdist,
                                               unsigned int method =␣
↪OENearestNbrsMethod::Default)
OESystem::OEIterBase<OENbrs>* OEGetNearestNbrs(const OEMolBase& mol,
                                               const double* coords,
                                               double maxdist,
                                               unsigned int method =␣
↪OENearestNbrsMethod::Default)
```

Returns an iterator over all atom pairs of the given molecule that are closer to each other than the given limit. When calculating the distances the coordinates passed as a parameter are utilized rather then the coordinates stored in the molecule.

maxdist
The maximum distance between two atoms.

method
Determines the algorithm that is used to find the nearest neighbors. This value has to be from the *OENearestNbrsMethod* namespace.

```
OESystem::OEIterBase<OENbrs>* OEGetNearestNbrs(const OEMolBase& mol1,
                                               const OEMolBase& mol2,
                                               double maxdist,
                                               unsigned int method =␣
↪OENearestNbrsMethod::Default,
                                               const bool sorting = false)
```

Returns an iterator over all atom pairs of the two molecules that are closer to each other than the given limit. This function calculates **intra-molecular** distances.

maxdist
The maximum distance between two atoms.

method
Determines the algorithm that is used to find the nearest neighbors. This value has to be from the *OENearestNbrsMethod* namespace.

sorting
If the value is set to `true`, the returned atom pairs are sorted based on the distance from the shortest to the longest. The default value is `false`.

See also

- *OENearestNbrs* function
- *OENbrs* class

OEGetNucleicAcidCode

```
char OEGetNucleicAcidCode(unsigned int idx)
```

Returns the single character nucleic acid code corresponding to the specified residue (using the *OEResidueIndex* namespace format). The character `N` is returned for the residue *OEResidueIndex.UNK* and the NULL character (0) is returned for all other non-nucleic acid residues.

See also

- *OEGetResidueIndexFromCode* inverse function
- *OEResidueIndex* namespace

OEGetMMCIFData

```
std::string OEGetMMCIFData(const OEMolBase& mol)
```

Returns the mmCIF metadata for a molecule. If the molecule does not have the data, an empty string is returned.

See also

- *mmCIF Metadata Manipulation* section

OEGetOEGraphMol

```
OEGraphMol OEGetOEGraphMol(OEBase obj, String tag)
```

Returns a copy of the *OEGraphMol* attached with the `tag` as *generic data*.

See also

- The *Attaching other objects* section for example code
- *OESetOEGraphMol*

OEGetOEMol

```
OEMol OEGetOEMol(OEBase obj, String tag)
```

Returns a copy of the *OEMol* attached with the `tag` as *generic data*.

See also

- *OESetOEMol*
- The *Attaching other objects* section for example code

OEGetOEQMol

```
OEQMol OEGetOEQMol(OEBase obj, String tag)
```

Returns a copy of the *OEQMol* attached with the `tag` as *generic data*.

See also

- The *Attaching other objects* section for example code
- *OESetOEQMol*

OEGetOtherNeighbor

```
OEBondBase* OEGetOtherNeighbor(const OEAtomBase* atm,
                               const OEBondBase* nbr)
OEAtomBase* OEGetOtherNeighbor(const OEAtomBase* atm,
                               const OEAtomBase* nbr)
```

OEGetPackedCoords

```
void OEGetPackedCoords(const OEMolBase& mol, float* coords)
void OEGetPackedCoords(const OEMolBase& mol, double* coords)
```

Fills an array with the coordinates of a molecule. The coordinates will be packed into an array of size *OEMolBase.NumAtoms* * 3 which is passed in as the coords array. The coordinates of the atoms will be in the array in the same order as the atoms come out of the iterator generated by *OEMolBase.GetAtoms*.

mol
The molecule to get coordinates from.

coords
The array to fill coordinates with.

OEGetPathLength

```
unsigned int OEGetPathLength(const OEAtomBase* a, const OEAtomBase* b,
                             unsigned int maxpath = 0)
```

Returns the shortest path length between two atoms i.e. the number of bonds between the two given atoms. For example, in thiazole (`c1nccs1`) the path length between the nitrogen and sulfur atom is 2.

src, dst
The two atoms for which the length of the shortest path returns. If the two atoms are the same *OEAtomBase*, have different parent molecule or are part of different disconnected molecular graphs, the *OEGetPathLength* function returns zero.

maxpath*
This parameter may be used to specify a maximum path length to consider before terminating the search. If this value is zero, the default, there is no upper bound on the returned path length. Otherwise, if the shortest path between the two atoms is greater than **maxpath** bonds, this function returns the value zero.

See also

- *OEShortestPath* function
- Example program *MinPath.cs*

OEGetPaulingVdWRadius

```
double OEGetPaulingVdWRadius(unsigned int elemno)
```

Returns the Van der Waals radius for a particular atomic number as tabulated by Linus Pauling [Pauling-1960]. There are several experimental values for Van der Waals radii (see also *OEGetBondiVdWRadius*) and many additional values used in molecular mechanics forcefields (including united-atom VdW radii), so there is often no definitive notion of Van der Waals radius.

OEGetPDBAtomIndex

```
unsigned int OEGetPDBAtomIndex(const char* name)
```

Returns an unsigned integer index encoding the atom name. The returned index is from the *OEPDBAtomName* namespace. If the PDB atom name is **not** recognized then the *OEGetPDBAtomIndex* function returns *OEPDBAtomName.UNK*.

```
unsigned int OEGetPDBAtomIndex(const OEAtomBase* atm)
```

Returns an unsigned integer index encoding the atom name that is returned by the *OEAtomBase.GetName* method. The returned index is from the *OEPDBAtomName* namespace.

Note

The PDB atom name index returned by the *OEGetPDBAtomIndex* function may be converted back into a string using the *OEGetPDBAtomName* function.

OEGetPDBAtomName

```
const char* OEGetPDBAtomName(unsigned int idx)
```

Converts an **OEChem TK** PDB atom name index, as returned by the *OEGetPDBAtomIndex* function, back into a string representation. Valid indices are specified in **OEChem TK**'s *OEPDBAtomName* namespace.

A NULL pointer, `(char*)0`, is returned for invalid or unrecognized argument values. All other returned string constants are guaranteed to be four characters long. For example, the input value *OEPDBAtomName.CA* returns the string value `" CA "`.

OEGetPDBBaseData

```
OESystem::OEIterBase<OESystem::OEBaseDataT<std::string> >* OEGetPDBBaseData(const␣
↪OESystem::OEBase& mol)
```

OEGetPDBDataPairs

```
OESystem::OEIterBase<OEPDBDataPair>* OEGetPDBDataPairs(const OEMolBase& mol)
```

Returns an iterator over all the PDB data pairs of the molecule.

See also

- *OEGetSDData* function
- *OEAddSDData* and *OESetSDData* functions
- *PDB Tagged Data Manipulation* section

OEGetPDBData

```
std::string OEGetPDBData(const OEMolBase& mol, const std::string& tag)
```

Returns the PDB data for the given tag. If the molecule does not have that tag, an empty string is returned. If there are multiple items with the same tag, only the first data is returned.

See also

- *OEGetPDBDataPairs* function to access multiple items with the same tag
- *OEAddSDData* and *OESetSDData* functions
- *PDB Tagged Data Manipulation* section

OEGetRadiiTypeName

```
const char* OEGetRadiiTypeName(unsigned int radiiType)
```

Returns the name of the specified `radiiType`, from the *OERadiiType* namespace.

See also

- *OEAssignRadii*

OEGetRadiusOfGyration

```
bool OEGetRadiusOfGyration(double* radiusOfGyration,
                           const double* pmi, const OEMolBase& mol)
bool OEGetRadiusOfGyration(double* radiusOfGyration,
                           const double* pmi, const OEConfBase& mol)
bool OEGetRadiusOfGyration(double* radiusOfGyration,
                           const double* pmi, const OEMCMolBase& mol)
```

Calculates the radius of gyration of the given molecule, using the given PMI that has already been calculated with the function *OECalcPMI*. Units of the calculated radius are the same as those used to set the radius of the molecule, which need to be the same radius units used to calculate the PMI.

The function returns true if calculation is successful, false otherwise. If the molecule is not centered, or radius of gyration values are all 0, the function will return false,

Note that for an *OEMCMolBase*, the *OECalcPMI* only calculates the PMI for the active conformer, and therefore this function similarly uses only the mass of the active conformer to calculate the radius of gyration.

radiusOfGyration
: This double array will be populated by the radius of gyration in each dimension, X, Y, and Z.

pmi
: This is a pointer to the PMI of the given molecule in each dimension, in X, Y, and Z.

mol
: This is the molecule for which the PMI is calculated. It must have a set radius and set mass for the calculation to be valid.

Warning

These functions only work correctly with a molecule that has been centered using the function *OECenter*. If the molecule has not been centered, the function will return false.

See also

- *OECalcPMI* function to calculate principle moment of inertia.

The following code snippet demonstrates how to calculate the PMI and radius of gyration for a molecule.

```
using System;
using OpenEye.OEChem;
```

(continues on next page)

(continued from previous page)

```
public class CalcPMI
{
    public static int Main(string[] args)
    {
        foreach (string inp_filename in args)
        {
            OEGraphMol mol = new OEGraphMol();
            oemolistream ifs = new oemolistream(inp_filename);
            double[] pmi = new double[3];
            double[] rGyr = new double[3];
            while (OEChem.OEReadMolecule(ifs, mol))
            {
                OEChem.OECalcPMI(pmi, mol);
                Console.WriteLine("PMI");
                foreach (double val in pmi)
                {
                    Console.Write(val + " ");
                }
                Console.WriteLine();
                OEChem.OEGetRadiusOfGyration(rGyr, pmi, mol);
                Console.WriteLine("Radius of Gyration");
                foreach (double val in rGyr)
                {
                    Console.Write(val + " ");
                }
                Console.WriteLine();
            }
        }
        return 0;
    }
}
```

OEGetResidueIndex

```
unsigned int OEGetResidueIndex(const char* name)
unsigned int OEGetResidueIndex(const OEResidue& res)
unsigned int OEGetResidueIndex(const OEAtomBase* atm)
```

Returns the **OEChem TK** PDB residue name index. The resulting value is specified by the *OEResidueIndex* namespace. If the residue name is not recognized internally by **OEChem TK** this function returns the value zero.

See also

- *OEResidueIndex* namespace
- *OEResidue* class
- *OEAtomGetResidue* function

OEGetResidueIndexFromCode

```
unsigned int OEGetResidueIndexFromCode(const char name)
```

Converts the single character amino acid code into a value from the *OEResidueIndex* namespace. For invalid amino acid codes the function returns `0`.

See also

- *OEGetAminoAcidCode* inverse function
- *OEResidueIndex* namespace

OEGetResidueName

```
const char* OEGetResidueName(unsigned int idx)
```

Converts an **OEChem TK** residue index, as returned by the *OEGetResidueIndex* function, back into a three-character string representation. Valid indices are specified in **OEChemTK**'s *OEResidueIndex* namespace.

A NULL pointer, `(char*)0`, is returned for invalid or unrecognized argument values. All other returned string constants are guaranteed to be three characters long. For example, the input value *OEResidueIndex.ALA* returns the string value `"ALA"`.

OEGetSDBaseData

```
OESystem::OEIterBase<OESystem::OEBaseDataT<std::string> >* OEGetSDBaseData(const␣
↪OESystem::OEBase& mol)
```

OEGetSDDataPairs

```
OESystem::OEIterBase<OESDDataPair>* OEGetSDDataPairs(const OEMolBase& mol)
```

Returns an iterator over all the SD data pairs of the molecule.

See also

- *OEGetSDData* function
- *OEAddSDData* and *OESetSDData* functions
- *SD Tagged Data Manipulation* section

Code Example

- *Adding csv data as SD tags* example

OEGetSDData

```
std::string OEGetSDData(const OEMolBase& mol, const std::string& tag)
```

Returns the SD data for the given tag. If the molecule does not have that tag, an empty string is returned. If there are multiple items with the same tag, only the first data is returned.

See also

- *OEGetSDDataPairs* function to access multiple items with the same tag
- *OEAddSDData* and *OESetSDData* functions
- *SD Tagged Data Manipulation* section

Code Example

- *Exporting SD data to a CSV file* example
- *Filter molecules by SD data* example

OEGetSingleBondNeighbor

```
OEAtomBase* OEGetSingleBondNeighbor(const OEAtomBase* atm)
```

OEGetSmallestSubtree

```
OESystem::OEIterBase<OEAtomBase>* OEGetSmallestSubtree(const OEBondBase * bond,
                                                       unsigned int maxdepth = 0)
OESystem::OEIterBase<OEAtomBase>* OEGetSmallestSubtree(const OEBondBase* bond,
                                                       const OESystem::OEUnaryPredicate
↪<OEAtomBase>& excludeAtoms,
                                                       unsigned int maxdepth = 0)
```

This function traverses the subgraph of all connected atoms on one end of a bond passed as the argument without crossing the bond itself. If the number of contiguous atoms connected to the beginning atom of the bond exceeds half of the atoms in the molecule, that subtree is discarded and a new traversal begins starting with the end atom of the bond. The bond itself may not be crossed in the second traversal. An iterator over the atoms that are members of the smallest subgraph is returned. The root atom of the bond from which the smallest subtree originates is included in the atom iterator returned.

maxdepth
: This parameter enables partial tree traversal in ring structures, default is zero which means full depth search.

See also

- *OEGetSubtree* function

OEGetSmiStringOrder

```
OESystem::OEIterBase<OEAtomBase>*OEGetSmiStringOrder(OEMolBase& mol,
                                                     unsigned int flavor =␣
↪OESMILESFlag::DEFAULT)
OESystem::OEIterBase<const OEAtomBase>*OEGetSmiStringOrder(const OEMolBase& mol,
                                                           unsigned int flavor =␣
↪OESMILESFlag::DEFAULT)
```

Returns an iterator over the heavy atoms of a molecule, in the order in which they appear in the canonical SMILES string. Several variants of SMILES format are supported by using different flavors from the *OESMILESFlag* namespace.

```
OEGraphMol mol = new OEGraphMol();
OEChem.OESmilesToMol(mol, "N1CCCC1");

foreach (OEAtomBase atom in OEChem.OEGetSmiStringOrder(mol))
{
    Console.WriteLine(atom.GetIdx() + " " + OEChem.OEGetAtomicSymbol(atom.
↪GetAtomicNum()));
}
```

See also

- *OECreateSmiString* function

OEGetSoleBond

```
OEBondBase* OEGetSoleBond(const OEAtomBase* atm)
```

Gets the first bond of the given atom. If there is more than one bond, the bond chosen is arbitrary. If there is no bond, null is returned.

OEGetSoleDoubleBond

```
OEBondBase* OEGetSoleDoubleBond(const OEAtomBase* atm)
```

Gets the first double bond of the given atom. If there is more than one double bond, the bond chosen is arbitrary. If there is no double bond, null is returned.

OEGetSoleNeighbor

```
OEAtomBase* OEGetSoleNeighbor(const OEAtomBase* atm)
```

Gets the first neighbor of the given atom. If there is more than one neighbor, the atom chosen is arbitrary. If there is no atom, the function returns a null pointer.

OEGetSoleSingleBond

```
OEBondBase* OEGetSoleSingleBond(const OEAtomBase* atm)
```

Gets the first single bond of the given atom. If there is more than one single bond, the bond chosen is arbitrary. If there is no single bond, null is returned.

OEGetSoleTripleBond

```
OEBondBase* OEGetSoleTripleBond(const OEAtomBase* atm)
```

Returns the first triple bond attached to the given atom. If there is more than one triple bond, the bond returned is arbitrary. if there is no triple bond, null is returned.

OEGetStyle

```
OE3DMolStyle OEGetStyle(const OEChem::OEMolBase& mol)
OE3DMolStyle OEGetStyle(const OEChem::OEAtomBase& atom)
OE3DMolStyle OEGetStyle(const OEChem::OEAtomBase& atom,
                        const OE3DMolStyle& dflt)
OE3DMolStyle OEGetStyle(const OEChem::OEAtomBase& atom,
                        const OEChem::OEConfBase& c)
```

Returns the *OE3DMolStyle* accociated with the molecule or atom specified. There are two various of this function that utilize *OEAtomBase* and either a *OE3DMolStyle* or *OEConfBase*. Both of these function are used to incorporate a conformer to influence the *OE3DMolStyle* returned.

OEGetStyleChange

```
OE3DMolStyle OEGetStyleChange(const OE3DMolStyle& initialStyle,
                              const OE3DMolStyle& finalStyle)
```

Returns a *OE3DMolStyle* with only the difference between the two inputted *OE3DMolStyles*. *OE3DMolStyle* &finalStyle is utilized as the reference *OE3DMolStyle*.

OEGetStyleIndex

```
unsigned int OEGetStyleIndex(const OEAtomBase& atom)
unsigned int OEGetStyleIndex(const OEConfBase& conf)
```

Returns the style index of a specified *OEAtomBase* or *OEConfBase* as an integer.

OEGetSubSearchScreenType

> **Attention**
>
> This API is currently available in C++ and Python.

```
const OESubSearchScreenTypeBase* OEGetSubSearchScreenType(const unsigned stype)
```

Converts a constant from the *OESubSearchScreenType* namespace into an internal screen type (*OESubSearchScreenTypeBase*).

```
const OESubSearchScreenTypeBase* OEGetSubSearchScreenType(const std::string& stypestr)
```

Convert a string (such as "Molecule SubSearch Screen,ver=0.9.0") into an internal screen type (*OESubSearchScreenTypeBase*). The function returns a null pointer for invalid strings.

See also

- *OESubSearchScreenTypeBase* class
- *OESubSearchScreenTypeBase.GetScreenTypeString* method
- *OESubSearchScreenType* namespace

Code Example

- *Generate database for fast substructure search* example
- *Perform SMARTS search on substructure database file* example

OEGetSubtree

```
OESystem::OEIterBase<OEAtomBase>* OEGetSubtree(const OEAtomBase* bgn,
                                               const OEAtomBase* end,
                                               unsigned int maxDepth = 0)
```

Traverses the subgraph from the **end** atom and avoids crossing over the bond to the **bgn** atom.

maxdepth
This parameter enables partial tree traversal in ring structures, default is zero which means full depth search.

```
OESystem::OEIterBase<OEAtomBase>* OEGetSubtree(const OEBondBase*,
                                               const OESystem::OEUnaryPredicate
↪<OEAtomBase>& excludeAtoms,
                                               unsigned int maxDepth = 0)
```

Traverses the subgraph from the beginning atom of the **bond** passed as the first argument without crossing the bond itself.

If the subgraph traversal does not cross any atoms which return `true` when tested with the **excludeAtoms** predicate, then an iterator over the atoms contained in the subgraph is returned. If ‘excludeAtoms` are encountered in the subgraph traversal, the subgraph is discarded and a new traversal begins from the end atom of the bond. If the new traversal fails to encounter excluded atoms then an iterator over the atoms contained in the subgraph is returned. If both traversals fail then an iterator over an empty set of atoms is returned.

maxdepth
This parameter enables partial tree traversal in ring structures, default is zero which means full depth search.

See also

- *OEGetSmallestSubtree* function

OEGetTokenizer

```
OETokenizerBase* OEGetTokenizer(unsigned int fmt, unsigned int flavor)
```

Returns an *OETokenizerBase* implementation for the specified `fmt` and `flavor` specified by constants in the *OEFormat* and *OEIFlavor* namespaces respectively. Tokenizers are the underlying implementation of how the *oemolithread* can chunk up a molecule file into a string token for each molecule. Most users will find it more convenient to use *oemolthreadbase.GetMol* method to retrieve molecule tokens as strings from a file.

OEGetTorsion

```
double OEGetTorsion(const OEMolBase& mol, const OEAtomBase* a,
                    const OEAtomBase* b, const OEAtomBase* c,
                    const OEAtomBase* d)
double OEGetTorsion(const OEConfBase& conf, const OEAtomBase* a,
                    const OEAtomBase* b, const OEAtomBase* c,
                    const OEAtomBase* d)
```

Returns the torsion formed by the four atoms *a*, *b*, *c*, and *d*. The return value is in radians and varies from `+PI` to `-PI`. If any of the arguments are null pointers, the function returns `FLT_MAX` to indicate an error. All of the atoms must be contained in the molecule (or conformer) which is passed to the function. The *OEConfBase* is strictly for efficiency. If the absolute value of the torsion can be used instead, the function *OEGetAbsTorsion* is much more efficient than calculating the absolute value of the return of this function.

mol
: The molecule that the atoms belong to.

conf
: The conformer that the atoms belong to.

a
: One of the end atoms in the torsion.

b
: Atom that is connected to *a* in the torsion.

c
: Atom that is connected to *d* in the torsion.

d
: The other end atom in the torsion.

See also

- *OEGetTorsion* **OEBio TK** function for regular dihedral angles in standard protein residues.

OEGetTorsions

```
OESystem::OEIterBase<OETorsion>* OEGetTorsions(const OEMolBase &,
                                               const OESystem::OEUnaryPredicate
↪<OEBondBase>& isRotor = static_cast<const OESystem::OEUnaryPredicate<OEBondBase>&> \␣
↪(IsRotor()))
```

Return an iterator over all torsions in the molecule as *OETorsion* objects. What bonds are considered torsions can be changed by passing a different predicate to the `isRotor` argument.

mol
The molecule to get torsions of.

isRotor
Predicate to define which bonds should be considered torsions.

OEGetTripleBondNeighbor

```
OEAtomBase* OEGetTripleBondNeighbor(const OEAtomBase* atm)
```

OEGetUniMolecularRxnIter

```
OESystem::OEIterBase<OEMolBase>* OEGetUniMolecularRxnIter(const OEMolBase& mol,
                                                          OEQMolBase& rxn,
                                                          bool strictSmirks = true,
                                                          unsigned maxMatches = 10000,
                                                          bool validateKekule = true)
OESystem::OEIterBase<OEMolBase>* OEGetUniMolecularRxnIter(const OEMolBase& mol,
                                                          OEQMolBase& rxn,
                                                          bool strictSmirks = true,
                                                          unsigned maxMatches = 10000,
                                                          bool validateKekule = true)
```

The first function overload returns an iterator over unique products from each individual application of the transformation provided by a *OEUniMolecularRxn* instance initialized with the function arguments.

The second function overload returns an iterator over unique products from each individual application of the transformation provided by a *OEUniMolecularRxn* instance initialized with the provided *OEUniMolecularRxnOptions* class.

See also

- *OEUniMolecularRxn* class
- *OEUniMolecularRxnOptions* class

OEHasAtomStereoHydrogens

```
bool OEHasAtomStereoHydrogens(const OEAtomBase* atm)
```

Returns true if the given atom has specified stereochemistry and no more than one hydrogen, deuterium or tritium isotope attached.

OEHasBondStereoHydrogens

```
bool OEHasBondStereoHydrogens(const OEAtomBase* atm)
```

Returns true if there is a double bond adjacent to the atom that has stereochemistry specified, and either endpoint of this bond has no more than one hydrogen, deuterium or tritium isotope attached.

OEHasDoubleBondO

```
bool OEHasDoubleBondO(const OEAtomBase* atm)
```

OEHasDoubleBond

```
bool OEHasDoubleBond(const OEAtomBase* atm)
```

Determines whether an atom is connected by one or more double bonds.

OEHasExplicitHydrogens

```
bool OEHasExplicitHydrogens(const OEMolBase& mol)
```

Determines whether a molecule has any explicit hydrogens. A return value of `true` indicates that one or more of the *OEAtomBase* 's of the given molecule has an atomic number of one (*OEElemNo.H*).

OEHasExplicitHydrogenExpression

```
bool OEHasExplicitHydrogenExpression(const OEQMolBase& qmol)
```

Returns *true* if the query molecule (*OEQMolBase*) has any explicit hydrogen(s). For example, if the query molecule has been initialized from the `CO[H]` or `[CO[#1,#6]]` SMARTS patterns.

See also

- *OEHasImplicitHydrogenExpression* function
- *SMARTS Pattern Matching* chapter

OEHasImplicitHydrogens

```
bool OEHasImplicitHydrogens(const OEMolBase& mol)
```

Determines whether a molecule as any implicit hydrogens. A return value of `true` indicates that one or more of the *OEAtomBase* 's of the given molecule has an 'implicit hydrogen count' property greater than or equal to one.

OEHasImplicitHydrogenExpression

```
bool OEHasImplicitHydrogenExpression(const OEQMolBase& qmol)
```

Returns *true* if the query molecule (*OEQMolBase*) has any implicit hydrogen(s). For example, if the query molecule has been initialized from the `[Ch1]` SMARTS pattern.

See also

- *OEHasExplicitHydrogenExpression* function
- *SMARTS Pattern Matching* chapter

OEHasMMCIFData

```
bool OEHasMMCIFData(const OEMolBase& mol)
```

Determines whether a molecule has mmCIF metadata stored on it.

See also

- *mmCIF Metadata Manipulation* section

OEHasMolDBIdx

```
bool OEHasMolDBIdx(const OESystem::OEBase& mol)
```

Returns whether `mol` came from a *OEMolDatabase* and would a positive value from *OEGetMolDBIdx*.

OEHasMultipleBond

```
bool OEHasMultipleBond(const OEAtomBase* atm)
```

Determines whether an atom is connected by any bonds with whose bond order, as returned by *OEBondBase.GetOrder*, is greater than one.

OEHasPartialCharges

```
bool OEHasPartialCharges(const OEMolBase& mol)
```

Returns `true` if any of the atoms of a molecule have a non-zero partial charge.

OEHasPDBData

```
bool OEHasPDBData(const OEMolBase& mol)
```

Determines whether a molecule has any PDB data.

```
bool OEHasPDBData(const OEMolBase& mol, const std::string& tag)
```

Determines whether a molecule has PDB data with a given tag.

See also

- *PDB Tagged Data Manipulation* section

OEHasRadii

```
bool OEHasRadii(const OEMolBase &mol, bool all=true)
```

OEHasRadii returns whether atoms in the molecule have radii assigned.

When `all` is `true` (the default), the function returns `true` only if every non-dummy atom has a non-zero radius. When `all` is `false`, the function returns `true` if any non-dummy atom has a non-zero radius.

Note

Dummy atoms (atomic number *OEElemNo.Du*) are excluded from the check.

See also

- *OEAssignRadii*
- *OEAtomBase.GetRadius*
- *Atom Radius Functions* section

OEHasResidue

```
bool OEHasResidue(const OEAtomBase* atm)
```

Checks whether an atom has any biopolymer residue information associated with it.

See also

- *OEResidue* class
- *OEAtomGetResidue* function
- *OEAtomSetResidue* function

OEHasResidues

```
bool OEHasResidues(const OEMolBase& mol)
```

Checks whether any of the atoms of a molecule have any biopolymer residue information associated with them.

OEHasSDData

```
bool OEHasSDData(const OEMolBase& mol)
```

Determines whether a molecule has any SD data.

```
bool OEHasSDData(const OEMolBase& mol, const std::string& tag)
```

Determines whether a molecule has SD data with a given tag.

See also

- *SD Tagged Data Manipulation* section

Code Example

- *Renaming molecules by SD field* example

- *Filter molecules by SD data* example

OEHasSingleBond

```
bool OEHasSingleBond(const OEAtomBase* atm)
```

Determines whether an atom is connected by one or more single bonds. This function returns `true` if the atoms implicit hydrogen count is non-zero.

OEHasStereoHydrogens

```
bool OEHasStereoHydrogens(const OEAtomBase* atm)
```

Returns whether the heavy atom passed in the argument has a hydrogen attached that is required to properly specify a stereo-center.

OEHasStyle

```
bool OEHasStyle(const OEChem::OEMolBase& mol)
```

Returns a boolean that indicates whether the *OEMolBase* has a *OE3DMolStyle*.

OEHasTokenizer

```
bool OEHasTokenizer(unsigned int fmt)
```

OEHasTripleBond

```
bool OEHasTripleBond(const OEAtomBase* atm)
```

Determines whether an atom is connected by one or more triple bonds.

OEHistogramFormula

```
void OEHistogramFormula(std::string& str, const unsigned int* hist,
                        int charge)
```

OEInChIToMol

```
bool OEInChIToMol(OEMolBase& mol, const std::string& str)
```

Converts an InChI string into a molecule. After constructing the molecule the following properties are perceived:

- Rings are perceived by calling the *OEFindRingAtomsAndBonds* function.
- The aromaticity is assigned by calling the *OEAssignAromaticFlags* function using the *OEAroModel.OpenEye* aromaticity model.
- The chirality is perceived by calling the *OEPerceiveChiral* function.

The molecule object is cleared before parsing so the resulting molecule object only contains atoms and bonds from the InChI. If one wishes to *add* atoms and bonds to an existing molecule, one can use the lower-level *OEParseInChI* function, which preserves any existing atoms and bonds in the molecule.

See also

- *OEMolToInChI* function
- *OEMolToSTDInChI* function
- *InChI* section

OEInit2DRingDictionary

```
bool OEInit2DRingDictionary(oemolistream& ims)
bool OEInit2DRingDictionary(const std::string& filename)
```

Initializes the **global** ring dictionary with the ring layouts stored in the given file. After initialization, the user-defined ring layouts will have priority over the built-in ring templates of **OEChem TK** when calling the *OEGenerate2DCoordinates* function to generate 2D coordinates.

See also

- *OE2DRingDictionary* class
- *OEIsValid2DRingDictionary* function
- *OEClear2DRingDictionary* function

2D ring dictionary files can be generated with the following programs:

- Example program *CreateRingDict.cs*
- Example program *AppendRingDict.cs*

The program below shows how to generate 2D coordinates with user-defined ring layouts:

- Example program *Generate2D.cs*

OEInitDefaultHandler

```
bool OEInitDefaultHandler(OESystem::OEBinaryIOHandlerBase& hand,
                          bool compressMCMol = false,
                          bool useZstd = false)
```

This function is equivalent to calling *OEInitHandler* with the return value of *OEBDefaultOpts* as both arguments.

OEInitHandler

```
bool OEInitHandler(OESystem::OEBinaryIOHandlerBase& hand,
                   const char* readOptHex, const char* writeOptHex)
```

Initialize the *OEBinaryIOHandlerBase* with all the handlers specified by the options `readOptHex` and `writeOptHex` for reading and writing OEB files. The hex strings passed in as arguments specify which handlers should be used. The hex strings should be the return value of one of the following functions:

- *OEBDefaultOpts*
- *OEBGZipCompressOpts*
- *OEBRotCompressOpts*

- *OEBAtomIntTypeOpts*

OEInvertCenter

```
bool OEInvertCenter(OEMolBase& mol, const OEAtomBase* atom)
bool OEInvertCenter(OEMolBase& mol, const OEAtomBase* atom,
                    const OEAtomBase* ref1, const OEAtomBase* ref2)
```

These functions perform a geometric inversion around the atom passed as the second argument. The second overloaded function allows the reference atoms that are held fixed during the inversion to be specified as the third and fourth arguments. The functions return `true` if inversion completes successfully. The functions return `false` if the center is not either degree three or degree four, if the reference atoms are invalid, if the center to invert has more than two rings bonds, or if a geometric inversion is ill defined.

mol
The molecule on which to perform inversion.

atom
The atom around which to perform the geometric inversion.

ref1
The first reference atom to be held fixed.

ref1
The second reference atom to be held fixed.

OEIs2DFormat

```
bool OEIs2DFormat(unsigned int format)
```

Returns `true` if the specified file format can contain 2D coordinates. The parameter value should be drawn from the *the OEFormat namespace*.

OEIs3DFormat

```
bool OEIs3DFormat(unsigned int format)
```

Returns `true` if the specified file format can contain 3D coordinates. The parameter value should be drawn from the *the OEFormat namespace*.

See also

- Example program *SMARTSAlign.cs*

OEIsBinary

```
bool OEIsBinary(unsigned int format)
```

Returns `true` if the specified file format is a binary format. The parameter value should be drawn from the *OEFormat* namespace.

OEIsCIFHeaderCompliant

Checks if mmCIF header metadata stored on the molecule is compliant with the input dictionary of supported CIF categories. The latest version of this dictionary can be found on the RCSB user guide

See also

- *OEGetMMCIFData* function to get mmCIF data
- *OESetMMCIFData* function to add mmCIF data
- *mmCIF Metadata Manipulation* section

OEIsCommonIsotope

```
bool OEIsCommonIsotope(unsigned int elemno, unsigned int mass)
```

Determine whether the isotope specified by the atomic number ('elemno'), and atomic mass ('mass') is a valid combination. This list is derived from high-energy physics experiments, and is useful for checking whether the masses provided on radioisotopes are within the expected range.

OEIsComplementaryScreenType

Attention

This API is currently available in C++ and Python.

```
bool OEIsComplementaryScreenType(const OEQMolBase& qmol,
                                 const unsigned stype)
bool OEIsComplementaryScreenType(const OEQMolBase& qmol,
                                 const OESubSearchScreenTypeBase* stype)
```

Returns whether the given query molecule can be used with a certain screen type. The function determines this by examining all the atom and bond expressions of the query molecule. For more explanation, see the *OESubSearchScreenType* namespace.

qmol
The query molecule (*OEQMolBase*)

stype
The type of the screen (either from the *OESubSearchScreenType* namespace or as an internal *OESubSearchScreenTypeBase* type).

See also

- *OESubSearchScreenType* namespace
- *OESubSearchScreenTypeBase* class

OEIsExtension

```
bool OEIsExtension(const char* fname)
```

OEIsGZip

```
bool OEIsGZip(const char *)
```

OEIsReadable

```
bool OEIsReadable(unsigned int format)
bool OEIsReadable(const std::string& filename)
bool OEIsReadable(unsigned int format, unsigned int flavor)
bool OEIsReadable(const std::string& filename, unsigned int flavor)
```

Returns `true` if the supplied file format is readable by **OEChem TK** via *OEReadMolecule*. The 'filename' can be a file name or a file extension. The 'format' parameter value should be drawn from the *OEFormat* namespace.

See also

- *OEReadMolecule* function

OEIsSDDataFormat

```
bool OEIsSDDataFormat(unsigned int format)
```

Returns `true` if the specified file format can contain SD data. The parameter value should be drawn from the *OEFormat namespace*.

See also

- *SD Tagged Data Manipulation* section

OEIsTerminalOxygen

```
bool OEIsTerminalOxygen(const OEAtomBase* atm)
```

OEIsTerminalSulfur

```
bool OEIsTerminalSulfur(const OEAtomBase* atm)
```

OEIsValid2DRingDictionary

```
bool OEIsValid2DRingDictionary(oemolistream& ifs)
bool OEIsValid2DRingDictionary(const std::string& filename)
```

Returns true is the given file is a valid 2D ring dictionary.

See also

- *OE2DRingDictionary* class
- *OEInit2DRingDictionary* class

2D ring dictionary files can be generated with the following programs:

- Example program *CreateRingDict.cs*
- Example program *AppendRingDict.cs*

OEIsValidSubSearchDatabase

Attention

This API is currently available in C++ and Python.

```
bool OEIsValidSubSearchDatabase(oemolistream& ifs)
bool OEIsValidSubSearchDatabase(const std::string& filename)
```

Returns `true` if the given molecule stream or the file is a valid substructure search database that was created by the *OECreateSubSearchDatabaseFile* function.

```
bool OEIsValidSubSearchDatabase(oemolistream& ifs,
                                const OESubSearchScreenTypeBase *)
bool OEIsValidSubSearchDatabase(const std::string& filename,
                                const OESubSearchScreenTypeBase *)
```

Returns `true` if the given molecule stream or the file is a valid substructure search database that was created by the *OECreateSubSearchDatabaseFile* function. These functions also check if the database stores substructure search screens with the given type.

See also

- *OESubSearchScreenTypeBase* class

Code Example

- *Perform SMARTS search on substructure database file* example

OEIsValidSubSearchScreenString

Attention

This API is currently available in C++ and Python.

```
bool OEIsValidSubSearchScreenString(const std::string& stypestr)
```

Returns `true` if the given string is a valid representation of any available screen type.

See also

- *OEGetSubSearchScreenType* function

OEIsWriteable

```
bool OEIsWriteable(unsigned int format)
bool OEIsWriteable(const std::string& filename)
bool OEIsWriteable(unsigned int format, unsigned int flavor)
bool OEIsWriteable(const std::string& filename, unsigned int flavor)
```

Returns `true` if the supplied file format is writable by **OEChem TK** via *OEWriteMolecule*. The 'filename' can be a file name or a file extension. The 'format' parameter value should be drawn from the *OEFormat* namespace.

See also

- *OEWriteMolecule* function

OEKekulize

```
bool OEKekulize(OEMolBase &)
```

Determine a valid Kekulé form for a molecule. On input, the integer bond types of each bond should be set to `1` for single, `2` for double, `3` for triple, `4` for quadruple and `5` for an aromatic (or resonant bond). On output, the bond order property of each bond is set to the integer bond types, except for bonds marked type `5`, which are assigned as either single or double as required. The formal charges and implicit hydrogen counts of each atom must be correctly specified prior to calling this function. The *OEKekulize* function returns false and throws a warning if the integer bond types are not set correctly. Newly created bonds will have their integer types set to `0` by default. These will need to be set to the appropriate values before calling *OEKekulize*; see the *Kekulé Assignment of New Bonds* section for examples.

Note that this function will not clear the aromatic flags associated with atoms and bonds. These flags may affect resulting output (*e.g.* SMILES) or downstream processing. So, if these effects are not desired, the function *OEClearAromaticFlags* should be used.

See also

- Example program *CanSmi.cs*

OEMacroModelAtomTypeNames

```
bool OEMacroModelAtomTypeNames(OEMolBase& mol)
```

This function sets the string atom type property of every atom in a molecule to its MacroModel atom type. The string atom type property is set using the *OEAtomBase.SetType* method. This function returns `false` if any atom was assigned `00`, *i.e.* unknown, and returns the value `true` otherwise.

OEMacroModelAtomTypes

```
bool OEMacroModelAtomTypes(OEMolBase& mol)
```

This function sets the integer atom type property of every atom in a molecule to its MacroModel atom type. The integer atom type indices are defined in the *OEMModType* namespace, and each atom is assigned a value using the *OEAtomBase.SetIntType* method. The *OEMacroModelAtomTypes* function returns `false` if any atom was assigned type zero, *i.e.* unknown, and returns the value `true` otherwise.

OEMacroModelTypeElement

```
unsigned int OEMacroModelTypeElement(unsigned int type)
```

Returns the atomic number/element represented by the given MacroModel atom type index, as encoded by the *OEMModType* namespace.

OEMacroModelTypeName

```
const char* OEMacroModelTypeName(unsigned int type)
```

This function returns the symbolic atom type name that corresponds to the given MacroModel atom type index, as encoded by the *OEMModType* namespace.

OEMacroModelTypeNames

```
bool OEMacroModelTypeNames(OEMolBase& mol)
```

This function loops through the atoms of the molecule, setting the string atom type property of each atom, to atom type name of the MacroModel atom type encoded in the atoms integer atom type property. This is approximately equivalent to following operation:

```
atm->SetType(OEMacroModelTypeName(atm->GetIntType()))
```

If both the string and integer atom types are required, it is faster to call *OEMacroModelAtomTypes* and then *OEMacroModelTypeNames* (which reuses the results of the first call) than it is to call *OEMacroModelAtomTypes* and *OEMacroModelAtomTypeNames*.

OEMakeSubSearchQueryScreen

Attention

This API is currently available in C++ and Python.

```
bool OEMakeSubSearchQueryScreen(OESubSearchScreen& screen,
                                OEQMolBase& qmol,
                                const OESubSearchScreenTypeBase* stype)
bool OEMakeSubSearchQueryScreen(OESubSearchScreen& screen,
                                OEQMolBase& qmol,
                                const unsigned stype = OESubSearchScreenType::Default)
```

Generates a substructure search screen of a query molecule with the given type.

screen
: The *OESubSearchScreen* object that is generated.

qmol
: The query molecule of which properties are encoded into the generated screen.

stype
: The type of the screen (either from the *OESubSearchScreenType* namespace or an internal screen type, *OESubSearchScreenTypeBase*).

The *OEMakeSubSearchQueryScreen* function also prepares the query molecule for the substructure search. For some screen types, the expression trees of the query are optimized for substructure search. The *OEMakeSubSearchQueryScreen* function returns `true` if the screen was successfully generated and `false` otherwise.

Note

For performance reason, **OEChem TK** does not support screening-based fast substructure search with explicit hydrogens. The *OEMakeSubSearchQueryScreen* function will return `false` if the query molecule has any explicit hydrogen expression.

OEIsComplementaryScreenType function can be used to check whether the query molecule is suited for a specific screen type.

See also

- *OEMakeSubSearchTargetScreen* function
- *OEScreen* function

OEMakeSubSearchTargetScreen

Attention

This API is currently available in C++ and Python.

```
bool OEMakeSubSearchTargetScreen(OESubSearchScreen& screen,
                                 OEMolBase& mol,
                                 const OESubSearchScreenTypeBase* stype)
bool OEMakeSubSearchTargetScreen(OESubSearchScreen& screen,
                                 OEMolBase& mol,
                                 const unsigned stype = OESubSearchScreenType::Default)
```

Generates a substructure search screen of a molecule with the given type.

screen
The *OESubSearchScreen* object that is generated.

mol
The molecule of which properties are encoded into the generated screen.

stype
The type of the screen (either from the *OESubSearchScreenType* namespace or an internal screen type, *OESubSearchScreenTypeBase*).

The *OEMakeSubSearchTargetScreen* function also prepares the molecule for the substructure search by doing the following.

- suppresses explicit hydrogens
- perceives the atom and bond properties of the molecule that are necessary to successfully execute the substructure search (see also the *OEPrepareSearch* function)

The *OEMakeSubSearchTargetScreen* function returns `true` if the screen was successfully generated and `false` otherwise.

See also

- *OEMakeSubSearchTargetScreen* function
- *OEScreen* function

OEMDLClearBondStereo

```
bool OEMDLClearBondStereo(OEMolBase& mol)
```

Clears the bond stereo fields from all bonds of a molecule.

See also

- *OEMDLPerceiveBondStereo* function
- *OEBondStereo*

OEMDLClearParity

```
void OEMDLClearParity(OEMolBase& mol)
void OEMDLClearParity(OEMCMolBase& mol)
```

Clears the MDL chiral flag, and atom stereo parity fields from all atoms of a molecule.

The *OEMCMolBase* overload of this function clears the MDL chiral flag on **all** conformers in the multi-conformer molecule.

See also

- *OEMDLPerceiveParity* function
- *OEMDLHasParity* function
- *OEMDLSetParity* and *OEMDLGetParity* functions

OEMDLCorrectBondStereo

```
bool OEMDLCorrectBondStereo(OEMolBase& mol)
```

This function tries to correct the wedge/hash bond assignment of the molecule, inverting or removing wedges and hashes from the connection table to avoid inconsistent or ambiguous stereochemistry assignment. This function requires that the molecule have 2D coordinates. See example in *Figure: Example of using OEMDLCorrectBondStereo*

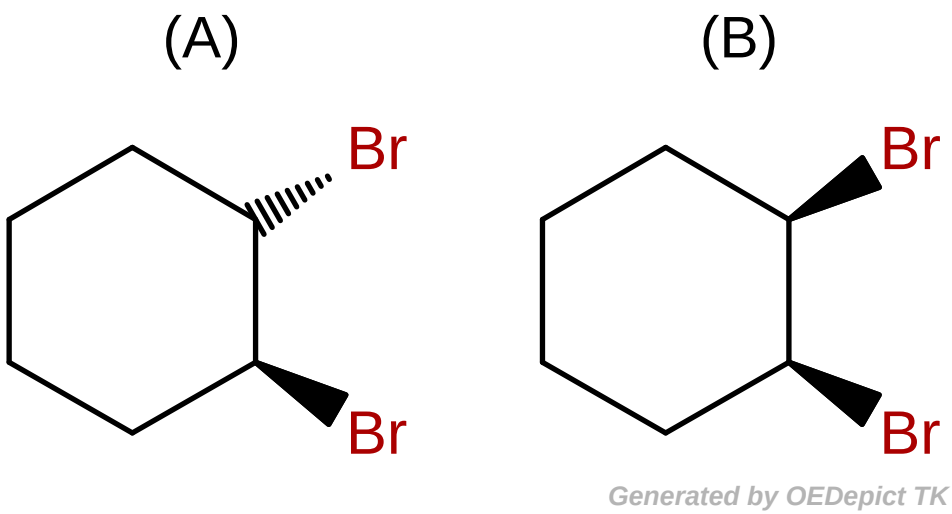


Fig. 10: **Example of using OEMDLCorrectBondStereo: before (A) and after (B) calling the OEMDLCorrectBondStereo function**

OEMDLGetParity

```
bool OEMDLGetParity(const OEMolBase& mol)
bool OEMDLGetParity(const OEMCMolBase& mol)
```

Returns the MDL 'chiral' flag status for a molecule. After reading from an MDL connection table this value is `true` if the chiral flag was 'on' in the header, indicating absolute stereochemistry for marked stereocenters. This value is

`false` if the chiral flag setting was 'off' in the header, indicating relative stereochemistry for marked stereocenters. After calling the *OEMDLPerceiveParity* function this value is forced to `true` if any non-zero atom stereo parity values have been assigned. The function *OEMDLHasParity* returns whether the 'chiral' flag status is available, which may indicate the structure originated from an MDL structure file. If *OEMDLHasParity* is `false`, this function will also return `false` by default.

The *OEMCMolBase* overload of this function returns the 'chiral' flag status of the **active** conformer in the multi-conformer molecule.

See also

- *OEMDLPerceiveParity* function
- *OEMDLHasParity* and *OEMDLSetParity* functions
- *OEMDLClearParity* function

OEMDLGetValence

```
unsigned int OEMDLGetValence(unsigned int atomicnum, int charge,
                             unsigned int minvalence)
```

Returns the **MDL valence** for the given parameters.

elemno
The atomic number returned by the *OEAtomBase.GetAtomicNum* method.

charge
The formal charge returned by the *OEAtomBase.GetFormalCharge* method.

minvalence
The minimum valence.

For example, *OEMDLGetValence* will return 4 in the case of the following input parameters:

- atomic number: *OEElemNo.N*
- formal charge: "+1"
- minimum valence: three single bonds to heavy atoms (value returned by the *OEAtomBase.GetExplicitDegree* method)

This means that the atom has correct valence only if it has either one implicit or one explicit hydrogen.

See also

- *MDL Valence Model* section
- *OEDefaultMDLHCount* function
- *OEAssignMDLHydrogens* function
- *OEIsValidMDLAtomValence* predicate

OEMDLHasIncorrectBondStereo

```
bool OEMDLHasIncorrectBondStereo(const OEMolBase& mol)
```

Checks whether the wedge/hash bond assignment of the molecule is inconsistent or ambiguous. This function requires that the molecule have 2D coordinates.

OEMDLHasParity

```
bool OEMDLHasParity(const OEMolBase& mol)
bool OEMDLHasParity(const OEMCMolBase& mol)
```

Checks for the availability of the MDL 'chiral' flag for a molecule. After reading from an MDL connection table this value is `true`, and `false` if the molecule did not originate from an MDL file or if the 'chiral' flag has been cleared. Calling the *OEMDLGetParity* function will return the actual on/off value of the 'chiral' flag from the MDL file header. After calling the *OEMDLClearParity* this function will return `false`.

The *OEMCMolBase* overload of this function checks the 'chiral' flag availability for the **active** conformer in the multi-conformer molecule.

See also

- *OEMDLPerceiveParity* function
- *OEMDLSetParity* and *OEMDLGetParity* functions
- *OEMDLClearParity* function

OEMDLPerceiveBondStereo

```
bool OEMDLPerceiveBondStereo(OEMolBase& mol)
```

Assigns wedge and hash bonds to a connection table from the **OEChem TK** stereochemistry of each atom. This function requires that the molecule have 2D coordinates. See example in *Figure: Example of using OEMDLPerceiveBondStereo*

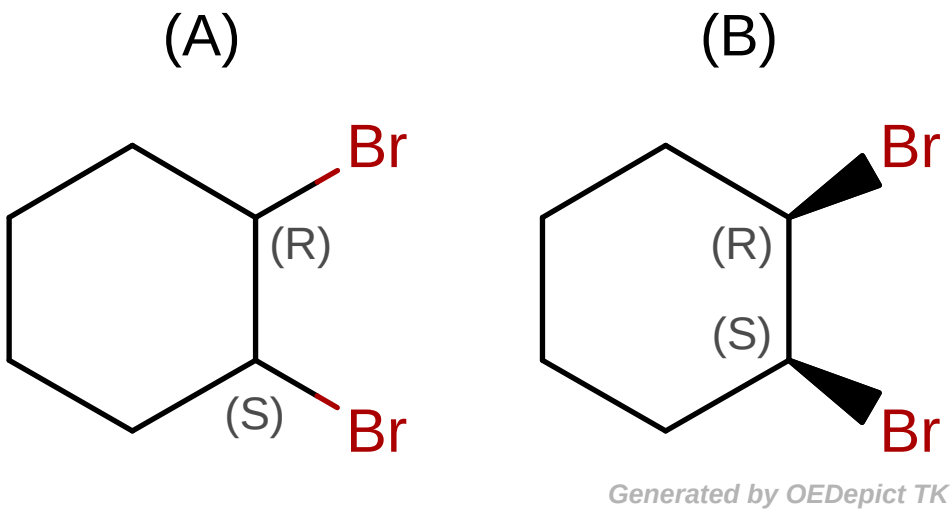


Fig. 11: **Example of using OEMDLPerceiveBondStereo: before (A) and after (B) calling the OEMDLPerceiveBondStereo function**

Note

This function is the opposite of the *OEMDLStereoFromBondStereo* function.

See also

- *OEBondStereo* namespace

Note

The *OEMDLPerceiveBondStereo* function preserves wavy bonds (*OEBondStereo.Wavy*). These can only be removed by calling the *OEMDLClearBondStereo* function.

OEMDLPerceiveParity

```
void OEMDLPerceiveParity(OEMolBase& mol)
```

This function sets the MDL stereo parity information for each atom from the **OEChem TK** stereochemistry. Any atom that has tetrahedral stereo specified has the corresponding MDL stereo parity value, one or two, placed in its 'MDLParity' generic data, with type `unsigned int`. This MDL stereo parity is relative to the current order of atoms returned by an iterator of atoms across a molecule. All other atoms, non-chiral or without stereo specified, have their MDL stereo parity set to zero or three.

OEMDLPerceiveParity internally calls the *OEPerceiveChiral* function, so there's no need to explicitly call *OEPerceiveChiral* on a molecule prior to calling *OEMDLPerceiveParity*.

Note

This function is the opposite of the *OEMDLStereoFromParity* function.

Note

This function assumes that atoms have stereo parity values assigned explicitly. This is generally automatic when reading structure file formats, or via generation of conformers. For 3D structures from other sources, an explicit call to the *OE3DToInternalStereo* function prior to calling this function may be required to ensure atom stereo parities are assigned based on the 3D coordinates.

See also

- *OEMDLHasParity* function
- *OEMDLSetParity* and *OEMDLGetParity* functions
- *OEMDLClearParity* function
- *OE3DToInternalStereo* function
- *OE3DToAtomStereo* function

OEMDLSetParity

```
bool OEMDLSetParity(OEMolBase& mol, bool value)
bool OEMDLSetParity(OEMCMolBase& mol, bool value)
```

Sets the MDL 'chiral' flag status for a molecule to the passed value. Setting the value to `true` implies setting the chiral flag 'on' and indicates absolute stereochemistry for any marked stereocenters. *OEMDLHasParity* will return true after setting a value via this function, and *OEMDLGetParity* will return the value. *OEMDLClearParity* will clear the property from the molecule.

The *OEMCMolBase* overload of this function sets the 'chiral' flag status of the **active** conformer in the multi-conformer molecule.

See also

- *OEMDLPerceiveParity* function

- *OEMDLHasParity* and *OEMDLGetParity* functions
- *OEMDLClearParity* function

OEMDLStereoFromBondStereo

```
bool OEMDLStereoFromBondStereo(OEMolBase& mol,
                               bool alterBondMarks = false)
```

Sets the **OEChem TK** stereochemistry of each atom from the wedge and hash bonds in the connection table. See example in *Figure: Example of using default OEMDLStereoFromBondStereo*

mol

The molecule of which stereochemistry is being set. The given molecule has to have 2D coordinates.

alterBondMarks

The boolean argument controls whether an attempt should be made to repair stereocenters that throw warnings due to ambiguous bond marking patterns (disabled by default with alterBondMarks=``false``). If this action is enabled, warnings will continue to be issued to allow user inspection and verification of the modified bond marks. However, once the corrected structure(s) have been rewritten with any format that captures bond stereo-mark information, a re-read of the structure(s) should emit fewer stereocenter correction warnings. Enabling the bond mark alteration activity will aggressively attempt to repair ambiguous bond marking patterns by possibly applying more than one stereo mark, or marking ring bond(s). See example in *Figure: Example of enabling alterBondMarks OEMDLStereoFromBondStereo*

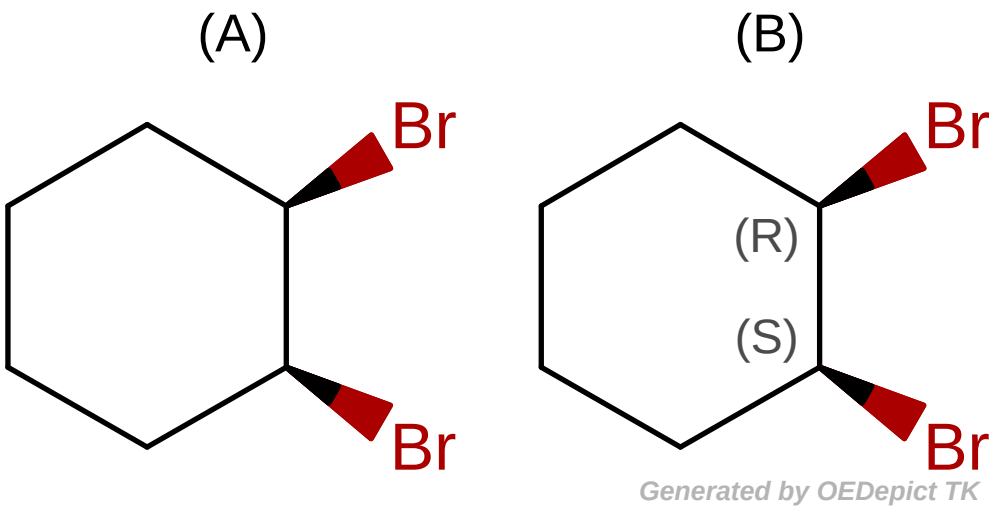


Fig. 12: **Example of using default OEMDLStereoFromBondStereo behavior: before (A) and after (B) calling the OEMDLStereoFromBondStereo function**

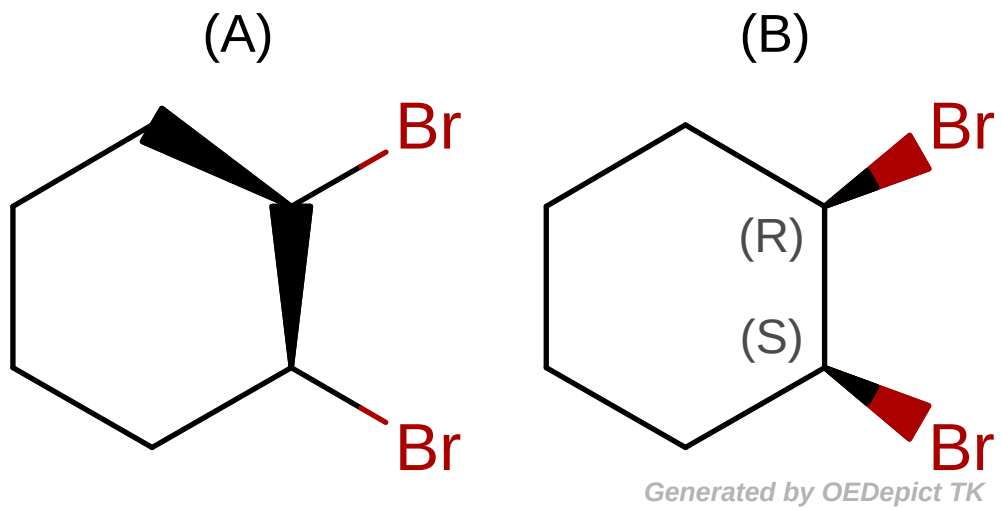


Fig. 13: **Example of enabling alterBondMarks OEMDLStereoFromBondStereo behavior: before (A) and after (B) calling the OEMDLStereoFromBondStereo function**

Note

This function is the opposite of the *OEMDLPerceiveBondStereo* function.

See also

- *Atom Stereo From Bond Stereo* chapter
- *OEAtomStereo* namespace
- *OEIFlavor.SDF.FixBondMarks* input flavor

OEMDLStereoFromParity

```
void OEMDLStereoFromParity(OEMolBase& mol)
```

This function sets the **OEChem TK** stereochemistry for each atom from the MDL stereo parity information.

Note

This function is the opposite of the *OEMDLPerceiveParity* function.

See also

- *OEMDLHasParity* function
- *OEMDLSetParity* and *OEMDLGetParity* functions
- *OEMDLClearParity* function

OEMMFF94InitialCharges

```
bool OEMMFF94InitialCharges(OEMolBase& mol)
```

Assigns integral or fractional formal atomic charges (for example +1/3 on guanidinium nitrogens), which are used in MMFF94 force field for obtaining final partial atomic charges applying bond charge increments. Requires a prior call of *OEMMFFAtomTypes*. Charge 0.0 is assigned to any atom missing proper MMFF94 atom type. Returns `true` when charges are successfully assigned.

Note

This function assigns silently zero charges to all atoms if MMFF94 atom types are not assigned to the passed molecule.

Hint

OEMMFF94InitialCharges is a low-level function, we highly recommend to assigning MMFF initial charges to a molecule by calling the `OEAssignCharges` function with the OEInitialCharges class of the **Quacpac TK**.

OEMMFF94PartialCharges

```
bool OEMMFF94PartialCharges(OEMolBase& mol)
```

Assigns MMFF94 partial charges. Requires a prior call of *OEMMFFAtomTypes*. Charge 0.0 is assigned to any atom missing proper MMFF94 atom type. Returns `true` when charges are successfully assigned.

Note

This function assigns silently zero charges to all atoms if MMFF94 atom types are not assigned to the passed molecule.

Hint

OEMMFF94PartialCharges is a low-level function, we highly recommend to assigning MMFF partial charges to a molecule by calling the `OEAssignCharges` function with the OEMMFF94Charges class of the **Quacpac TK**.

OEMMFFAtomTypeNames

```
bool OEMMFFAtomTypeNames(OEMolBase& mol)
```

Assigns symbolic MMFF94 atom names to atoms in passed molecule. Returns `true` when symbolic atom types are successfully assigned.

OEMMFFAtomType

```
unsigned int OEMMFFAtomType(const OEAtomBase* atom, bool& fail)
```

Returns the MMFF94 atom type from the *OEMMFFType* namespace.

The following table describes how the returned MMFF type and the 'fail' flag should be interpreted:

MMFF type	'fail' flag	interpretation
OEMMFFType.UNK	true	MMFF atom type **can not** be identified
not OEMMFFType.UNK	true	an **alternative** MMFF atom type is identified
not OEMMFFType.UNK	false	MMFF atom type is identified **successfully**

See also

OEMMFFType namespace

```
OEMMFFTypeResult OEMMFFAtomType(const OEAtomBase* atom)
```

Returns whether an MMFF94 integer atom type can be assigned to an atom and the type itself stored in an *OEMMFFTypeResult* object.

Warning

Only the function that returns *OEMMFFTypeResult* is supported in the wrapped languages: Python, Java and C#.

```
OEGraphMol mol = new OEGraphMol();
OEChem.OESmilesToMol(mol, "c1ccccc1C[Se]");

foreach (OEAtomBase atom in mol.GetAtoms())
{
    OEMMFFTypeResult mmffres = new OEMMFFTypeResult(OEChem.OEMMFFAtomType(atom));
    Console.WriteLine(atom.GetIdx() + " " + mmffres.GetFailed() + " " +
                      OEChem.OEMMFFTypeName(mmffres.GetMMFFType()));
}
```

The correct atom typing requires the following conditions:

- The molecule has to have explicit hydrogens.
- The molecule has to have (*MMFF*) aromaticity model.

Hint

We highly recommend assigning MMFF partial charges to a molecule by calling the `OEAssignCharges` function with the OEMMFF94Charges class of the **Quacpac TK**.

OEMMFFAtomTypes

```
bool OEMMFFAtomTypes(OEMolBase& mol)
```

Assigns MMFF94 integer atom type to atoms in passed molecule. Returns `true` when all atom types are successfully assigned.

The correct atom typing requires the following conditions:

- The molecule has to have explicit hydrogens.
- The molecule has to have (*MMFF*) aromaticity model.

Hint

We highly recommend assigning MMFF partial charges to a molecule by calling the `OEAssignCharges` function with the OEMMFF94Charges class of the **Quacpac TK**.

OEMMFFClearRemappedElements

```
void OEMMFFClearRemappedElements()
```

Removes any mappings created by calling *OEMMFFRemapElement*.

Warning

This function is **not** thread-safe. It should only be called when it is assured not to race with any other OEMMFF functions.

OEMMFFRemapElement

```
void OEMMFFRemapElement(unsigned int from_element,
                        unsigned int to_element)
```

Map a particular element to another element for purposes of the OEMMFF functions.

Warning

This function is **not** thread-safe. It should only be called when it is assured not to race with any other OEMMFF functions.

OEMMFFTypeIndex

```
unsigned int OEMMFFTypeIndex(unsigned int type)
```

OEMMFFTypeName

```
const char* OEMMFFTypeName(unsigned int type)
```

OEMMFFTypeNames

```
bool OEMMFFTypeNames(OEMolBase& mol)
```

OEMolecularFormula

```
void OEMolecularFormula(std::string& str, const OEMolBase& mol,
                        bool hydrogenisotopic = false)
```

This function calculates the molecular formula for the given molecule, return the result in the str argument. The component elements of the molecular formula are listed in Hill order. This function determines the cumulative molecular formula, so elements from different connected components are summed. To generate a "dot disconnected" molecular formula, this function should be used to generate the formula for each component or part, and the results concatenated together as appropriate. If the 'hydrogenisotopic' parameter is `true`, and the molecule contains any atoms of deuterium or tritium, then the output molecular formula will represent them as D or T as appropriate. By default hydrogens isotopes are summed together as H.

OEMolToInChIKey

```
std::string OEMolToInChIKey(const OEMolBase& mol)
```

Creates an InChIKey from the nonstandard round-trip InChI from a molecule.

> **Note**
>
> The underlying InChI code is v1.06 released on December 15, 2020.

OEMolToInChI

```
std::string OEMolToInChI(const OEMolBase& mol)
```

Creates a nonstandard round-trip form of InChI from a molecule. This function suppresses most of the normalizations done for the standard InChI to allow its use for round-trip activities.

> **Note**
>
> The underlying InChI code is v1.06 released on December 15, 2020.

Table 39: Success rate for **OEMolTo[STD]InChI/OEInChIToMol** to regenerate input structure `SMILES`

Database	Size	Number of failures	Success Rate
ChEMBL_23 (OEMolToSTDInChI)	1.7M	~798K	53 %
ChEMBL_23 (OEMolToInChI)	1.7M	~92K	95 %

The round-trip failures relate to *InChI* structure normalization activities (charge, and/or bond rearrangements) that preclude the ability to recover an arbitrary molecule in its input form.

> **See also**
>
> - *OEMolToSTDInChI* function
> - *OEInChIToMol* function
> - *OECreateInChI* function
> - *OEParseInChI* function

OEMolToCXSMILES

```
std::string OEMolToCXSMILES(const OEMolBase& mol)
```

Creates a CXSMILES from a molecule by evoking the low-level *OECreateSmiString* function with the *OESMILESFlag.ISOMERIC* and *OESMILESFlag.EnhStereo* options.

Prior to generating the CXSMILES the following properties are perceived **if necessary**:

- The rings of the molecule are perceived by calling the *OEFindRingAtomsAndBonds* function
- The aromaticity is assigned by calling the *OEAssignAromaticFlags* function using the *OEAroModel.OpenEye* aromaticity model.
- In case of a 3D molecule, the stereochemistry of the atoms and the bonds are set by calling the *OE3DToAtomStereo* and the *OE3DToBondStereo* functions, respectively.

See also

- *OECXSMILESToMol* function
- *OESmilesToMol* function
- *Unique Representation* section

OEMolToSmiles

```
std::string OEMolToSmiles(const OEMolBase& mol)
```

Creates a canonical isomeric SMILES from a molecule by evoking the low-level *OECreateSmiString* function with the *OESMILESFlag.ISOMERIC* option.

Prior to generating the canonical isomeric SMILES the following properties are perceived **if necessary**:

- The rings of the molecule are perceived by calling the *OEFindRingAtomsAndBonds* function
- The aromaticity is assigned by calling the *OEAssignAromaticFlags* function using the *OEAroModel.OpenEye* aromaticity model.
- In case of a 3D molecule, the stereochemistry of the atoms and the bonds are set by calling the *OE3DToAtomStereo* and the *OE3DToBondStereo* functions, respectively.

See also

- *OESmilesToMol* function
- *OECXSMILESToMol* function
- *Unique Representation* section

OEMolToSTDInChIKey

```
std::string OEMolToSTDInChIKey(const OEMolBase& mol)
```

Creates an InChIKey from the a standard InChI for a molecule.

Note

The underlying InChI code is v1.06 released on December 15, 2020.

OEMolToSTDInChI

```
std::string OEMolToSTDInChI(const OEMolBase& mol)
```

Creates a standard InChI from a molecule.

Note

The underlying InChI code is v1.06 released on December 15, 2020.

Table 40: Success rate for **OEMolTo[STD]InChI/OEInChIToMol** to regenerate input structure SMILES

Database	Size	Number of failures	Success Rate
ChEMBL_23 (OEMolToSTDInChI)	1.7M	~798K	53 %
ChEMBL_23 (OEMolToInChI)	1.7M	~92K	95 %

The round-trip failures relate to *InChI* structure normalization activities (charge, and/or bond rearrangements) that preclude the ability to recover an arbitrary molecule in its input form.

See also

- *OEMolToInChI* function
- *OEInChIToMol* function
- *OECreateInChI* function
- *OEParseInChI* function

OEMultipleBondCount

```
unsigned int OEMultipleBondCount(const OEAtomBase* atm)
```

Returns the number of bonds connected to an atom, with bond order greater than one. The bond order is as determined by *OEBondBase.GetOrder*.

OENetCharge

```
int OENetCharge(const OEMolBase& mol)
```

Determines the net charge on a molecule. If the molecule has specified partial charges, see **OEChem TK**'s *OEHasPartialCharges* function, this function returns the sum of the partial charges rounded to an integer. Otherwise this function returns the sum of the formal charges on each atom of the molecule.

OENewMCMolBase

```
OEMCMolBase* OENewMCMolBase(unsigned int type = OEMCMolType::OEDefault)
OEMCMolBase* OENewMCMolBase(const OEMolBase& mol,
                            unsigned int type = OEMCMolType::OEDefault)
OEMCMolBase* OENewMCMolBase(const OEMCMolBase& mol,
                            unsigned int type = OEMCMolType::OEDefault)
OEMCMolBase* OENewMCMolBase(const OESystem::OEBase& b,
                            unsigned int type = OEMCMolType::OEDefault)
```

The *OENewMCMolBase* factory functions allocate an implementation of an *OEMCMolBase* based on the integer 'type' argument. The 'type' argument specifies the implementation type to be allocated, and should be one of the values listed in *OEMCMolType* namespace. An *OEMolBase* or *OEMCMolBase* may be passed in order to copy construct the *OEMCMolBase* instance. The pointer to the *OEMCMolBase* memory returned by the function is allocated dynamically. The `delete` operator must be called for all returned instances to prevent memory leaks.

OENewMolBase

```
OEMolBase* OENewMolBase(unsigned int type = OEMolBaseType::OEDefault)
OEMolBase* OENewMolBase(const OEMolBase& mol,
                        unsigned int type = OEMolBaseType::OEDefault)
```

Allocates an implementation of an *OEMolBase* based on the integer 'type' argument. The 'type' argument specifies the implementation type to be allocated, and should be one of the values listed in *OEMolBaseType*. The returned molecule may be copy constructed from an existing *OEMolBase* instance. The pointer to the *OEMolBase* memory returned by the function is allocated dynamically. The `delete` operator must be called for all returned instances to prevent memory leaks.

OENewQMolBase

```
OEQMolBase* OENewQMolBase(unsigned int type = OEQMolType::OEDefault)
OEQMolBase* OENewQMolBase(const OEMolBase& mol,
                          unsigned int type = OEQMolType::OEDefault)
OEQMolBase* OENewQMolBase(const OEQMolBase& mol,
                          unsigned int type = OEQMolType::OEDefault)
```

Allocates an implementation of an *OEQMolBase* based on the integer 'type' argument. The 'type' argument specifies the implementation type to be allocated, and should be one of the values listed in *OEQMolType*. An *OEMolBase* or *OEQMolBase* may be passed in order to copy construct the *OEQMolBase* instance. The pointer to the *OEQMolBase* memory returned by the function is allocated dynamically. The `delete` operator must be called for all returned instances to prevent memory leaks.

OENormalize

```
void OENormalize(OEMolBase& mol,
                 const OENormModel model = OENormModelDefault)
```

OEOverlayConfsByRMSD

```
bool OEOverlayConfsByRMSD(OEMCMolBase& mol,
                          const unsigned maxTermHeavyAtoms = 25u)
```

This method overlays the all conformers of the molecule, onto the first conformer, such that the RSMD between the conformers are minimized. The automorphisms of the molecule graph is taken into consideration during the RMSD

calculations.

mol
Molecule whose conformers are to be overlaid.

maxTermHeavyAtoms
Maximum number of terminal heavy atoms that should be considered in assessing automorphisms in the molecule graphs. If the number of terminal heavy atoms exceeds this limit, they are ignored for assessing automorphisms.

OEParseInChI

```
bool OEParseInChI(OEMolBase& mol, const std::string& str)
```

Parses a InChI string into a molecule. The returned value indicates whether the given InChI string was parsed successfully.

mol
The constructed molecule.

str
The InChI string that is parsed into the molecule.

The input molecule instance is not cleared so these functions can be used to add components to an existing molecule.

Hint

The use of the high-level *OEInChIToMol* function is highly recommended over the low-level *OEParseInChI* function. The *OEInChIToMol* function not only converts an InChI string into a molecule but also perceives basic molecular properties.

See also

- *OEMolToSTDInChI* function
- *OEMolToInChI* function
- *OEInChIToMol* function
- *OECreateInChI* function
- *InChI* section

OEParseSmarts

```
bool OEParseSmarts(OEQMolBase& mol, const char* smarts,
                   unsigned int opt = OESmartsParseOpts::Default)
bool OEParseSmarts(OEQMolBase& mol, const char* smarts,
                   const OEVectorBindings& vbinding,
                   unsigned int opt = OESmartsParseOpts::Default)
```

These functions generate a query molecule by parsing the given SMARTS pattern. If the SMARTS string passed to the function is valid and is parsed correctly the function will return `true`, otherwise it will return `false`.

mol
The *OEQMolBase* object that is generated for the given SMARTS pattern.

smarts
The SMARTS string that represents a molecular substructure query.

opts
The parameter determines how the SMARTS pattern is interpreted. This value has to be from the *OESmartsParseOpts* namespace.

vbinding
The parameter provides a mechanism for parsing a SMARTS pattern that contains vector bindings. An *OEVectorBindings* object containing the complete set of possible vector bindings that may be contained in a SMARTS pattern, including recursive vector bindings, and passed to the *OEParseSmarts* function. Vector bindings will be resolved while the SMARTS pattern is being parsed.

See also

- *SMARTS Pattern Matching* chapter
- *OESmartsParseOpts* namespace

OEParseSmiles

```
bool OEParseSmiles(OEMolBase& mol, const char* str,
                   bool canon = false, bool strict = false)
bool OEParseSmiles(OEMolBase& mol, const std::string& str,
                   bool canon = false, bool strict = false)
bool OEParseSmiles(OEMolBase& mol, const char* str,
                   const OEParseSmilesOptions& options)
bool OEParseSmiles(OEMolBase& mol, const std::string& str,
                   const OEParseSmilesOptions& options)
```

Parses a SMILES string into a molecule. The returned value indicates whether the given SMILES string was parsed successfully.

mol
The constructed molecule.

str
The SMILES string that is parsed into the molecule.

canon
Flag that determines whether the post-processing kekulization is performed. Passing a boolean `true` value to this argument indicates to the parser that the SMILES string should be assumed to be well-formed and the usual kekulization (by calling *OEKekulize*) step may be omitted.

This can be used to speed-up parsing of a large database, but has the side-effect that bond orders are not correctly assigned for aromatic molecules.

strict
Flag that controls whether the parser should operate in strict mode. By default, the SMILES parser attempts to process any reasonably formed SMILES string. If the 'strict' argument is `true`, the parser applies more rigorous sanity checking. For example, the SMILES `C==C` is accepted by the default non-strict parser, but rejected by the strict parser.

options
The *OEParseSmilesOptions* object that stores properties that influence how a SMILES string is parsed into a molecule.

The input molecule instance is not cleared so these functions can be used to add components to an existing molecule. If the SMILES string represents a reaction, any previously-existing atoms and bonds not already in a reaction component become reactants in the resulting molecule object.

Hint

The use of the high-level *OESmilesToMol* function is highly recommended over the low-level *OEParseSmiles* function. The *OESmilesToMol* function not only converts a SMILES string into a molecule but also perceives basic molecular properties.

See also

- *OESmilesToMol* function
- *Unique Representation* section

OEParseSmirks

```
bool OEParseSmirks(OEQMolBase& mol, const char* smirks,
                   unsigned int opt = OESmartsParseOpts::Default)
bool OEParseSmirks(OEQMolBase& mo, const char* smirks,
                   const OEVectorBindings& vb,
                   unsigned int opt = OESmartsParseOpts::Default)
```

These functions generate a reaction molecule by parsing the given SMIRKS pattern. If the SMIRKS string passed to the function is valid and is parsed correctly the function will return a `true`, otherwise the it will return `false`.

mol
: The *OEQMolBase* object that is generated for the given SMIRKS pattern.

smirks
: The SMARTS string that represents a molecular substructure query.

opts
: The parameter determines how the SMARTS pattern is interpreted. This value has to be from the *OESmartsParseOpts* namespace.

vbinding
: The parameter provides a mechanism for parsing a SMARTS pattern that contains vector bindings. An *OEVectorBindings* object containing the complete set of possible vector bindings that may be contained in a SMIRKS pattern, including recursive vector bindings, and passed to the *OEParseSmarts* function. Vector bindings will be resolved while the SMIRKS pattern is being parsed.

See also

- *SMARTS Pattern Matching* chapter
- *OESmartsParseOpts* namespace

Note

OEParseSmirks is functionally nearly identical to *OEParseSmarts*. The only significant difference in how SMARTS and SMIRKS are parsed is the handling of the [H] expression. SMARTS interprets [H] as an atom that has a total hydrogen count of exactly one. SMIRKS interprets [H] as an explicit hydrogen. The *OEParseSmirks* function therefore parses [H] differently than would *OEParseSmarts*, and performs additional validity checks of the SMIRKS string.

See also

- Example program *LibGen.cs*

OEPDBOrderAtoms

```
bool OEPDBOrderAtoms(OEMolBase& mol, bool preserveResOrder = false)
```

Reorders the atoms of a molecule into PDB order, using the residue information associated with each atom. The optional argument preserveResOrder, forces the function to preserve the residue order from the input.

OEPerceiveBondOrders

```
void OEPerceiveBondOrders(OEMolBase& mol)
```

Attempts to determine bond orders from the 3D geometry and connectivity of a molecule. If the connectivity is unspecified, the *OEChem* function *OEDetermineConnectivity* must be called prior to this function.

This function invalidates the aromaticity flags of the molecule's atoms and bonds. If necessary, these aromaticity flags may either be cleared using *OEClearAromaticFlags* or set appropriately by calling *OEAssignAromaticFlags*, after calling this function.

See also

- *Connectivity Perception* chapter and *Bond Order Perception* section

OEPerceiveChiral

```
bool OEPerceiveChiral(OEMolBase& mol, bool carbonOnly = false)
```

Perceives whether an atom or a bond in a molecule is chiral and updates the 'chiral' property accordingly. Because this function is based on the symmetry classes of a molecule, as a side-effect, it also sets the 'symmetry' property of all the atoms the molecule to be their 'explicit hydrogen suppressed' symmetry class, by calling the **OEChem TK** function *OEPerceiveSymmetry*. The default behavior is to perceive nitrogens that can undergo nitrogen inversion as chiral centers. This behavior can be turned off with the optional carbonOnly flag. Before calling OEPerceiveChiral with a different flag it is important to first clear the existing chiral perception using *OEClearChiralPerception*.

See also

- *OEClearChiralPerception* to clear existing chiral perception

Note

The aromaticity (see *Aromaticity Perception* chapter) has to be perceived by calling the *OEAssignAromaticFlags* function prior to calling *OEPerceiveCIPStereo*.

OEPerceiveCIPStereo

```
unsigned int OEPerceiveCIPStereo(const OEMolBase& mol,
                                 const OEAtomBase* atom)
```

Perceives the Cahn-Ingold-Prelog descriptor for the given atom. The return value is taken from the *OECIPAtomStereo* namespace.

For a chiral atom center the function returns

- either *OECIPAtomStereo.R* or *OECIPAtomStereo.S*, if the atom has specified stereochemistry
- *OECIPAtomStereo.UnspecStereo* if the CIP atom stereo center has no specified stereo (*i.e.* when *OEAtomBase.HasStereoSpecified* returns false),

The function always returns *OECIPAtomStereo.NotStereo* for the following atoms:

- non-chiral atoms
- atoms that have invalid valence (checked by *OECheckAtomValence*)

See also

- *OESetCIPStereo* to set atom stereo from CIP stereo descriptor
- *CIP Stereo Perception* section

```
unsigned int OEPerceiveCIPStereo(const OEMolBase& mol,
                                 const OEBondBase* bond)
```

Perceives the Cahn-Ingold-Prelog descriptor for the given bond stereo center. The return value is taken from the *OECIPBondStereo* namespace.

For a double bond, the function returns

- either *OECIPBondStereo.E* or *OECIPBondStereo.Z*, if the bond has specified stereochemistry
- *OECIPBondStereo.UnspecStereo* if the CIP bond stereo center has no specified stereo (*i.e.* when *OEBondBase.HasStereoSpecified* returns false)

For a non-chiral bond, the function always returns *OECIPBondStereo.NotStereo*.

See also

- *OESetCIPStereo* to set bond stereo from CIP stereo descriptor
- *CIP Stereo Perception* section
-

OEPerceiveResidues

```
void OEPerceiveResidues(OEMolBase& mol,
                        unsigned int stable = OEPreserveResInfo::None)
```

Recognizes the protein and nucleic acid chains of a molecule and set the residue information for each atom appropriately. The stable parameter is a bit-mask defined in the *OEPreserveResInfo* namespace that indicates if the residue numbers, residue names, or chain ID's should be preserved during the perception routine. By default, no prior information is preserved.

OEPerceiveSymmetry

```
bool OEPerceiveSymmetry(OEMolBase &, bool includeH = true,
                        bool automorph = false)
```

Set the structural symmetry class of each atom of the given molecule. Symmetry classes are numbered sequentially from one up to the maximum number of atoms in the molecule for totally asymmetric molecules. The assigned symmetry class for each atom may be retrieved using the *OEAtomBase.GetSymmetryClass* method.

The 'includeH' parameter is used to specify whether implicit hydrogens should be considered distinct from explicit hydrogens. Normally, a molecule will have all hydrogens either implicitly represented or explicitly represented so this option shouldn't affect symmetries.

OEPRECompress

Warning

OEB files generated through these functions will only be readable by **OEChem TK** 2.0.2 (2014.Oct) and higher.

```
bool OEPRECompress(oemolstreambase& str)
```

Enables the *oemolostream* to write rotor-offset-compressed molecules in the perfect-rotor-encoding format, even for `float` precision molecules. This alleviates the need to GZip the `OEB`, improving the performance of *OEMolDatabase.Open*.

Note

To compress data for **ROCS/FastROCS** at the *OEMCMolBase* level instead of *oemolostream* see `OEPrepareFastROCSMol`.

OEPrepareSearch

```
void OEPrepareSearch(OEMolBase& mol, const OESubSearch& subs,
                     bool adjustH = false)
```

Perceives the atom and bond properties of the given **target** molecule that are necessary to successfully execute the given substructure search.

- The *OEAssignAromaticFlags* function is called if the aromaticity of the molecule is not perceived yet. The aromaticity is also re-perceived if it does not match the aromaticity model of the query molecule stored in the given *OESubSearch* object. If the query does not have an aromatic model specified after processing (e.g., SMARTS, SMIRKS input), the target is re-perceived with the default *OEAroModel.OpenEye* aromaticity model.

- The *OEPerceiveChiral* function is called if the chirality of the given molecule is not perceived yet.
- The *OEAssignHybridization* function is called if the hybridization perception is required by the query molecule stored in the *OESubSearch* object.
- Attempts to perceive atom stereo from other information (such as 3D, bond stereo or atom parity) if the atom stereo is required by the query molecule stored in the *OESubSearch* object.
- Attempts to perceive bond stereo from 2D or 3D coordinates if the bond stereo is required by the query molecule stored in the *OESubSearch* object.
- If the **adjustH** parameter is `true`, and
  - the query molecule of the substructure search has any explicit hydrogen expressions but no implicit ones, then *OEAddExplicitHydrogens* is called on the given **target** molecule.
  - the query molecule of the substructure search has any implicit hydrogen expressions but no explicit ones, then *OESuppressHydrogens* is called on the given **target** molecule.

Warning

If the query molecule has both explicit and implicit hydrogens then the hydrogens on the **target** molecule can not be adjusted accordingly, and the substructure search could fail. Therefore it is highly recommended to **not mix** explicit and implicit hydrogens in the query molecule.

Hint

It is a good practice to call the *OEPrepareSearch* function before calling any of the following methods:

- *OESubSearch.AtomMatch*
- *OESubSearch.Match*
- *OESubSearch.SingleMatch*

If there is any atom or bond property that is not perceived but it is necessary to successfully execute the substructure search, then the above methods throw a warning. These warnings can be suppressed by calling the *OEPrepareSearch* function.

See also

- *OESubSearch* class
- *OEHasExplicitHydrogenExpression* function
- *OEHasImplicitHydrogenExpression* function

OEPreserveRotCompress

```
bool OEPreserveRotCompress(oemolstreambase& str)
```

Instructs the *oemolstreambase* to preserve rotor-offset-compression data on *OEMCMolBase* objects when being read from the `OEB` file format. This should be called on the *oemolistream* object if the molecule needs to be output and preserve compression.

> **Warning**
>
> Altering the coordinates or connection table of an *OEMCMolBase* molecule that has been read through this mechanism is undefined. Modifications to conformer coordinates will likely be lost. Modifications to the connection table will likely result in an error during *OEWriteMolecule*.

OERandomizeTorsions

```
bool OERandomizeTorsions(OEMolBase& mol, double maxRadians)
bool OERandomizeTorsions(OEMolBase& mol, double maxRadians,
                         const OESystem::OEUnaryPredicate<OEBondBase>& isRotor)
```

Modifies each of the torsions in a molecule by a random value between +`maxRadians` and -`maxRadians`. While most people think about this sort of function in degrees, all angles in **OEChem TK** are measures in radians. The following simple constants may be used *OEMath.Pi*, *OEMath.Rad2Deg* and *OEMath.Deg2Rad* to generate the appropriate value.

There are two API points, the one with two arguments applies the random torsion adjustment to every bond for which the **OEChem TK** bond predicate 'IsRotor` returns `true`. The second API point has an additional argument in which the user can specify a functor which defines which bonds are rotatable. If the user's functor causes the function to attempt to apply a random rotation on a ring bond, no change in the internal coordinates will occur, however, the frame of reference of the molecule may change.

The functions return `true` if completes successfully. The functions return `false` if there are no torsions to randomize.

mol
: The molecule to randomize torsions of.

maxRadians
: Maximum value of torsions angle to be modified, in radians.

isRotor
: Predicate to define which torsions should be randomized.

OEReadCDXFile

```
bool OEReadCDXFile(oemolistream& ifs, OEMolBase& mol,
                   unsigned int flavor = OEIFlavor::CDX::DEFAULT)
```

Reads the contents of a ChemDraw CDX file into the specified *OEMolBase*.

OEReadCIFFile

```
bool OEReadCIFFile(oemolistream& ifs, OEMolBase& mol,
                   unsigned int flavor)
```

Reads a molecule from the specified input stream, 'ifs', in CIF file format. THis supports the small molecule smCIF, macromolecule mmCIF, and chemical component ccCIF format variants. The reader peeks into the file to check for specific tags to determine whether it is a smCIF, ccCIF or mmCIF formatted file, given that the extension used is the same. The mmCIF reader can be enforced by setting the *OEFormat.MMCIF* on the input stream. A number of different format variants are supported by the use of 'flavor' parameter from the *OEIFlavor.CIF* or the *OEIFlavor.MMCIF* namespace. This function returns `true` if the operation was successful, and `false` if an end-of-file was encountered.

The atoms (_atom_site) table has four "author defined alternatives" (.auth_*) that have similar meaning to the "primary" identifiers (.label_*). Two of them, atom name (atom_id) and residue name (comp_id) almost never differ. The other two, chain name (asym_id) and sequence number (seq_id) may differ in a confusing way (A,B,C <-> C,A,B). Thus we read and store only one, auth if it is present, otherwise label.

In all PDB entries each auth_asym_id "chain" is split into one or more label_asym_id subchains. The polymer (residues before the TER record in the PDB format) goes into one subchain; all the other (non-polymer) residues are put into single-residue subchains; except the waters, which are all put into one subchain. Currently, wwPDB treats non-linear polymers (such as sugars) as non-polymers.

Note

It is recommended to always use the author-defined names, for consistency with the PDB format and with the literature.

OEReadCSVFile

```
bool OEReadCSVFile(oemolistream& ifs, OEMolBase& mol,
                   unsigned int flavor = OEIFlavor::CSV::DEFAULT)
```

Read the next molecule from a comma-separated-value, CSV, file opened by `ifs` into `mol`. The specific layout of the file is described in the *CSV File Format* section. The `flavor` argument is a value OR'd together from the values in the *OEIFlavor.CSV* namespace. Returns `true` if a molecule was successfully read, `false` otherwise.

OEReadCSVHeader

```
bool OEReadCSVHeader(oemolistream& ifs, OECSVHeader& header,
                     unsigned int flavor = OEIFlavor::CSV::DEFAULT)
```

Attempt to read the first line of the file opened by `ifs` into a *OECSVHeader* object. This initializes the *OECSVHeader* object with column names described by that line. If *OEIFlavor.CSV.Header* is not specified in the `flavor` argument or the first column of the first line is interpretable as SMILES, then the column names will be initialized to the following pattern based upon the number of columns found in the first line: SMILES, TITLE, OE_CSV_COLUMN_1, OE_CSV_COLUMN_2, ..., OE_CSV_COLUMN_N.

See also

CSV File Format

OEReadFASTAFile

```
bool OEReadFASTAFile(oemolistream& ifs, OEMolBase& mol,
                     bool customResidues = false,
                     bool embeddedSmiles = false)
```

Reads the contents of a FASTA sequence file into the specified *OEMolBase*. The amino acid residues of the sequence file are automatically expanded to an all atom representation.

OEReadMacroModelFile

```
bool OEReadMacroModelFile(oemolistream& ifs, OEMolBase& mol)
bool OEReadMacroModelFile(oemolistream& ifs, OEMCMolBase& mol)
```

Reads the contents of the Schrodinger MacroModel format file into the specified *OEMolBase*.

OEReadMDLFile

```
bool OEReadMDLFile(oemolistream& ifs, OEMolBase& mol,
                   const char* id = "")
```

Reads the contents of the MDL mol or SDF file into the specified *OEMolBase*.

OEReadMDLQueryFile

```
bool OEReadMDLQueryFile(oemolistream& ifs, OEMolBase& mol)
```

Reads the contents of the MDL query file into the specified *OEMolBase* object. The atom and bond query features present in the MDL file are attached as generic data to the related atoms and bonds of the *OEMolBase* object. An *OEQMolBase* object can be subsequently created from this *OEMolBase* object by calling the *OEBuildMDLQueryExpressions* function. *OEReadMDLQueryFile* is a low-level reader and as such will not call *Clear* on each read.

See also

- *Substructure Search with MDL Queries* chapter

```
bool OEReadMDLQueryFile(oemolistream& ifs, OEQMolBase& qmol,
                        unsigned int opts = OEMDLQueryOpts::Default)
```

Reads the contents of the MDL query file into the specified *OEQMolBase* object. The option from the *OEMDLQueryOpts* namespace determine how the MDL query is interpreted.

See also

- *Substructure Search with MDL Queries* chapter

OEReadMDLReactionQueryFile

```
bool OEReadMDLReactionQueryFile(oemolistream& ifs, OEMolBase& mol)
```

Reads the contents of the MDL reaction file into the specified *OEMolBase* object. *OEReadMDLReactionQueryFile* is a low-level reader and as such will not call *Clear* on each read.

```
bool OEReadMDLReactionQueryFile(oemolistream& ifs, OEQMolBase& qmol,
                                unsigned int opts = OEMDLQueryOpts::ReactionQuery)
```

Reads the contents of the MDL reaction file into the specified *OEQMolBase* object. The option from the *OEMDLQueryOpts* namespace determine how the MDL reaction is interpreted.

See also

- *MDL Reaction Query File* section
- Example program *LibGen.cs*

OEReadMol2File

```
bool OEReadMol2File(oemolistream& ifs, OEMolBase& mol,
                    bool m2h = false)
```

Reads the contents of the Tripos Sybyl mol2 file into the specified *OEMolBase*. The 'm2h' parameter indicates whether the connection table already has all hydrogen atoms explicit, informing the **OEChem TK** mol2 file reader not to adjust implicit hydrogen counts to satisfy valence and formal charge rules.

OEReadMolecule

```
bool OEReadMolecule(oemolistream& ifs, OEMol& mol)
bool OEReadMolecule(oemolistream& ifs, OEQMol& mol)
bool OEReadMolecule(oemolistream& ifs, OEMolBase& mol)
bool OEReadMolecule(oemolithread& ifs, OEMolBase& mol)
bool OEReadMolecule(oemolistream& ifs, OEGraphMol& mol)
bool OEReadMolecule(oemolithread& ifs, OEQMolBase& mol)
bool OEReadMolecule(oemolistream& ifs, OEQMolBase& mol)
bool OEReadMolecule(oemolistream& ifs, OEMCMolBase& mol)
bool OEReadMolecule(oemolithread& ifs, OEMCMolBase& mol)
```

Reads the contents of the input molstream into the specified molecule using the file format currently associated with the *oemolistream*. The assumed file format may be specified using the *oemolistream.SetFormat* method.

See also

- Example program *Convert.cs*

OEReadMolFromBytes

```
boolean OEReadMolFromByteArray(OEMolBase mol, String format,
                               OEIntArray data)
boolean OEReadMolFromByteArray(OEMCMolBase mol, String format,
                               OEIntArray data)

boolean OEReadMolFromBytes(OEMCMolBase mol, String format, byte[] data)
boolean OEReadMolFromBytes(OEMCMolBase mol, int format, int flavor,
                           boolean gzip, byte[] data)

boolean OEReadMolFromBytes(OEMolBase mol,  String format, byte[] data)
boolean OEReadMolFromBytes(OEMolBase mol, int format, int flavor,
                           boolean gzip, byte[] data)

boolean OEReadMolFromBytes(OEMCMolBase mol, String format, byte[] data)
boolean OEReadMolFromBytes(OEMCMolBase mol, int format, int flavor,
                           boolean gzip, byte[] data)
```

Instantiates a molecule from the contents of 'data' in terms of 'format'. 'format' must be a file extension that is readable by OpenEye, for example: ".oeb.gz". Overloads which take an unsigned 'format', unsigned 'flavor' and Boolean 'gzip' parameters expect the format or the format and flavor to be specified by one of the constants in *OEFormat* and *OEIFlavor*.

Returns 'true' if decoding was successful.

> **Warning**
>
> The overload that takes an OEIntArray is deprecated as of 2018.Feb and will be removed in a future version of **OEChem TK**, please use *OEReadMolFromBytes* instead.

```
OEMolBase mol = new OEGraphMol();
bool success = OEChem.OEReadMolFromBytes(mol, ".smi", Encoding.ASCII.GetBytes("c1ccccc1
↪"));
```

```
OEMolBase mol = new OEGraphMol();
bool success = OEChem.OEReadMolFromBytes(mol, OEFormat.SMI, OEChem.
↪OEGetDefaultIFlavor(OEFormat.SMI), false, Encoding.ASCII.GetBytes("c1ccccc1"));
```

OEReadMOPACFile

```
bool OEReadMOPACFile(oemolistream& ifs, OEMolBase& mol)
```

Reads the contents of the MOPAC output file into the specified *OEMolBase*.

OEReadOEBFile

```
unsigned int OEReadOEBFile(OEPlatform::oeistream& is, OEMolBase& mol,
                           const OESystem::OEBinaryIOHandlerBase& handler)
unsigned int OEReadOEBFile(OEPlatform::oeistream& is,
                           OEMCMolBase& mol,
                           const OESystem::OEBinaryIOHandlerBase& handler)
```

Reads a molecule from the specified input stream in the OpenEye Binary Version 2 format. This is considered a low-level API for advanced users that will only parse the part of the OEB specified by the handlers registered with *handler*. *OEReadMolecule* is the preferred way to read molecules from OEB files, as it will handle many corner cases more cleanly. This function only performs the parsing of the actual OEB data based upon the given 'handler'.

OEReadPDBFile

```
bool OEReadPDBFile(oemolistream& ifs, OEMolBase& mol,
                   unsigned int flavor = OEIFlavor::PDB::DEFAULT)
```

Reads a molecule from the specified input stream, 'ifs', in Brookhaven PDB file format. A number of different PDB format variants are supported by the use of 'flavor' parameter from the *OEIFlavor.PDB* namespace. This function returns `true` if the operation was successful, and `false` if an end-of-file was encountered.

OEReadRxnFile

```
bool OEReadRxnFile(oemolistream& ifs, OEMolBase& mol,
                   const char* id = "", bool query = false)
```

OEReadSketchFile

```
bool OEReadSketchFile(oemolistream& ifs, OEMolBase& mol)
```

Reads the contents of the MDL ISIS sketch file into the specified *OEMolBase*.

OEReadXYZFile

```
bool OEReadXYZFile(oemolistream& ifs, OEMolBase& mol)
```

Reads the contents of the XMol XYZ format file into the specified *OEMolBase*.

OEResidueHydrogens

```
void OEResidueHydrogens(OEAtomBase* atm,
                        unsigned style = OEPDBHydrogenStyle::Modern)
void OEResidueHydrogens(OEMolBase& mol,
                        unsigned style = OEPDBHydrogenStyle::Modern)
```

This function sets the residue information associated with each explicit hydrogen in a molecule, from the residue information of its parent.

OEResidueToString

```
std::string OEResidueToString(const OEChem::OEResidue& res,
                              const std::string delimiter = " ")
```

Returns the string representation of the given residue in the following format:

```
<RES NAME> <RES NUMBER> <INSERT CODE> <CHAIN ID> <FRAG NUMBER> <ALT>
```

Examples:

```
ASP 381   A 1
72L 301   A 5
THR 60 A B 2
THR 9   A 1 A
```

See also

- *OEResidue* class

OERGroupsToHydrogens

```
bool OERGroupsToHydrogens(OEMolBase& mol)
```

Converts terminal R-groups (atoms with atomic number 0) in a molecule to hydrogen atoms. The terminal requirement means it can only and must be be connected to one heavy atom. Additionally, it R-group atoms implicit hydrogen count must be zero. The function attempts to be consistent with the input molecule, such that molecules with only implicit hydrogens will stay that way after the conversion. If a molecule has explicit hydrogens, the function will convert the R-group to an explicit hydrogen atom. For a molecule in 3D, the new hydrogen will have 3D coordinates, either retaining the coordinates of the R-group atom, of that was places at (0,0,0) the hydrogen will be assigned proper 3D coordinates.

See also

- *OESet3DHydrogenGeom* that assigned proper 3D geometry to the hydrogen atom if needed.

OERingBondCount

```
unsigned int OERingBondCount(const OEChem::OEAtomBase&)
```

Returns the number of bonds connected to this atom that are part of a ring.

OERMSD

Array-Based OERMSD

```
double OERMSD(const float* refcrds, const float* fitcrds,
              unsigned int size, bool overlay = false,
              double* rot = 0, double* trans = 0)
double OERMSD(const double* refcrds, const double* fitcrds,
              unsigned int size, bool overlay = false,
              double* rot = 0, double* trans = 0)
```

Returns the root mean squared deviation (i.e. RMSD) between two sets of Cartesian coordinates.

refcrds, fitcrds
: These arrays should be of length *size* *3, and should contain the Cartesian coordinates of the two objects being assessed.

overlay
: This flag indicates whether the RMSD of the two arrays in their current position is desired (`false`), or whether the lowest possible RMSD for the two arrays should be returned (`true`).

rot, trans
: If an overlay calculation is carried out, the functions can report the rotation and translation required to achieve the minimum RMSD. An array of length `double[9]` should be passed to the **rot** argument and an array of length `double[3]` should be passed as the **trans** argument.

Example:

```
private static void OERMSD_Array(double[] refcrds, double[] fitcrds, uint size)
{
  double rmsd1 = OEChem.OERMSD(refcrds, fitcrds, size);

  bool overlay = true;
  double rmsd2 = OEChem.OERMSD(refcrds, fitcrds, size, overlay);

  double[] rotmat = new double[9];
  double rmsd3 = OEChem.OERMSD(refcrds, fitcrds, size, overlay, rotmat);

  double[] transvec = new double[3];
  double rmsd4 = OEChem.OERMSD(refcrds, fitcrds, size, overlay, rotmat, transvec);
}
```

Full Molecule-Based OERMSD

```
double OERMSD(const OEMolBase& ref, const OEMolBase& fit,
              bool automorph = true, bool heavyOnly = true,
              bool overlay = false, double* rot = 0, double* trans = 0)
bool OERMSD(const OEMolBase& ref, const OEMCMolBase& fit,
            double* rmsdArray, bool automorph = true,
            bool heavyOnly = true, bool overlay = false,
            double* rot = 0, double* trans = 0)
```

Calculates the root mean squared deviation (i.e. RMSD) between two molecules.

ref, fit
This function is overloaded for comparisons of a single-conformer (*OEMolBase*) reference molecule with either a single-conformer (*OEMolBase*) or a multi-conformer (*OEMCMolBases*) **fit** molecule.

rmsdArray
For the *OEMolBase* vs *OEMolBase* comparison, the RMSD is the return value. For the *OEMolBase* vs *OEMCMolBase* case, the RMSDs are returned in the **rmsdArray** array. The **rmsdArray** passed to this function should be of length `fit.GetMaxConfIdx()`.

automorph
This flag indicates whether automorphisms should be taken into account during the RMSD calculation. Automorphisms are the symmetry related transformations of a molecule which can result in anomalously high RMSDs if not properly treated. For instance, t-butyl-benzene has a three-fold automorphism around the t-butyl group and a two-fold automorphism around the benzene ring.

heavyOnly
This flag indicates whether only heavy atoms should be considered or hydrogen atoms should be also taken into account as well when assessing automorphisms.

Hint

It is strongly recommended that one should consider carefully before setting the 'automorph' flag to `true` and the 'heavyOnly' flag to `false` due to the increased computational cost.

overlay
This flag indicates whether the RMSD of the molecules in their current position is desired (`false`), or whether the lowest possible RMSD for the two molecules should be returned (`true`).

rot, trans
If an overlay calculation is carried out, the functions can report the rotation and translation required to give this minimum RMSD.

In case when the **fit** molecule is single-conformer (*OEMolBase*), array of length `double[9]` should be passed to the **rot** argument and an array of length `double[3]` should be passed as the **trans** argument.

In case when the **fit** molecule is multi-conformer (*OEMCMolBase*), the array size of both **rot** and **trans** has to be multiplied by the length of `fit.GetMaxConfIdx()`.

Hint

The arrays **rot** and **trans** described above can subsequently be applied to the **fit** molecule using the *OERotate* and *OETranslate* functions if desired. It is important that *OERotate* and *OETranslate* are applied in that order.

Note

A multi-conformer **fit** molecule (*OEMCMolBases*) that is passed to the *OERMSD* function can have nonsequential conformation indices. This can occur, for example, if some conformations of the molecule have been deleted. In this case, the returned **rmsdArray** array will contain `-1.0` RMSD values for invalid conformation indices.

Example:

```
private static void OERMSD_Full_MolBase(OEMolBase refMol, OEMolBase fitMol)
{
  double rmsd1 = OEChem.OERMSD(refMol, fitMol);

  bool automorf = true;
  double rmsd2 = OEChem.OERMSD(refMol, fitMol, automorf);

  bool heavyOnly = true;
  double rmsd3 = OEChem.OERMSD(refMol, fitMol, automorf, heavyOnly);

  bool overlay = true;
  double rmsd4 = OEChem.OERMSD(refMol, fitMol, automorf, heavyOnly, overlay);

  double[] rotmat = new double[9];
  double rmsd5 = OEChem.OERMSD(refMol, fitMol, automorf, heavyOnly, overlay, rotmat);

  double[] transvec = new double[3];
  double rmsd6 = OEChem.OERMSD(refMol, fitMol, automorf, heavyOnly, overlay, rotmat,␣
↪transvec);
}

private static void OERMSD_Full_MCMolBase(OEMolBase refMol, OEMCMolBase fitMol)
{
  uint nConfs = fitMol.GetMaxConfIdx();
  double[] vecRmsd = new double[nConfs];
  OEChem.OERMSD(refMol, fitMol, vecRmsd);

  bool automorf = true;
  OEChem.OERMSD(refMol, fitMol, vecRmsd, automorf);

  bool heavyOnly = true;
  OEChem.OERMSD(refMol, fitMol, vecRmsd, automorf, heavyOnly);

  bool overlay = true;
  OEChem.OERMSD(refMol, fitMol, vecRmsd, automorf, heavyOnly, overlay);

  double[] rotmat = new double[9*nConfs];
  OEChem.OERMSD(refMol, fitMol, vecRmsd, automorf, heavyOnly, overlay, rotmat);

  double[] transvec = new double[3*nConfs];
  OEChem.OERMSD(refMol, fitMol, vecRmsd, automorf, heavyOnly, overlay, rotmat, transvec);
}
```

Partial Molecule-Based OERMSD

```
double OERMSD(const OEMolBase& ref, const OEMolBase& fit,
              const OEMatchBase& match, bool overlay = false,
              double* rot = 0, double* trans = 0)
bool OERMSD(const OEMolBase& ref, const OEMCMolBase& fit,
            double* rmsdArray, const OEMatchBase& match,
            bool overlay = false, double* rot = 0, double* trans = 0)
```

These functions are quite similar to the previous three. However, rather than considering automorphisms and heavy atoms, these functions allow a user to explicitly determine which substructure of the two molecules should be used to

determine the RMSD.

ref, fit
: This function is overloaded for comparisons of a single-conformer (*OEMolBase*) reference molecule with either a single-conformer (*OEMolBase*) or a multi-conformer (*OEMCMolBases*) **fit** molecule.

rmsdArray
: For the *OEMolBase* vs *OEMolBase* comparison, the RMSD is the return value. For the *OEMolBase* vs *OEMCMolBase* case, the RMSDs are returned in the **rmsdArray** array. The **rmsdArray** passed to this function should be of length `fit.GetMaxConfIdx()`.

match
: The match determines the atoms of the **fit** molecule that will be aligned to the atoms of the *ref* molecule. The match can be generated by hand, or with any of the **OEChem TK** matching algorithms such as *OESubSearch* or *OEMCSSearch*.

overlay
: This flag indicates whether the RMSD of the molecules in their current position is desired (`false`), or whether the lowest possible RMSD for the two molecules should be returned (`true`).

rot, trans
: If an overlay calculation is carried out, the functions can report the rotation and translation required to give this minimum RMSD.

  In case when the **fit** molecule is single-conformer (*OEMolBase*), array of length `double[9]` should be passed to the 'rot' argument and an array of length `double[3]` should be passed as the 'trans' argument.

  In case when the **fit** molecule is multi-conformer (*OEMCMolBase*), the array size of both *rot* and *trans* has to be multiplied by the length of `fit.GetMaxConfIdx()`.

Hint

The arrays **rot** and **trans** described above can subsequently be applied to the **fit** molecule using the *OERotate* and *OETranslate* functions if desired. It is important that *OERotate* and *OETranslate* are applied in that order.

Example:

```
private static void OERMSD_Part_MolBase(OEMolBase refMol, OEMolBase fitMol)
{
  OEMatch match = new OEMatch();
  OEAtomBaseIter aref = refMol.GetAtoms();
  OEAtomBaseIter afit = fitMol.GetAtoms();
  while (aref.IsValid() && afit.IsValid())
  {
      match.AddPair(aref.Target(), afit.Target());
      aref.Increment();
      afit.Increment();
  }

  double rmsd1 = OEChem.OERMSD(refMol, fitMol, match);

  bool overlay = true;
  double rmsd2 = OEChem.OERMSD(refMol, fitMol, match, overlay);

  double[] rotmat = new double[9];
  double rmsd3 = OEChem.OERMSD(refMol, fitMol, match, overlay, rotmat);
```

(continues on next page)

(continued from previous page)

```
  double[] transvec = new double[3];
  double rmsd4 = OEChem.OERMSD(refMol, fitMol, match, overlay, rotmat, transvec);
}

private static void OERMSD_Part_MCMolBase(OEMolBase refMol, OEMCMolBase fitMol)
{
  OEMatch match = new OEMatch();
  OEAtomBaseIter aref = refMol.GetAtoms();
  OEAtomBaseIter afit = fitMol.GetAtoms();
  while (aref.IsValid() && afit.IsValid())
  {
      match.AddPair(aref.Target(), afit.Target());
      aref.Increment();
      afit.Increment();
  }

  uint nConfs = fitMol.GetMaxConfIdx();
  double[] vecRmsd = new double[nConfs];
  OEChem.OERMSD(refMol, fitMol, vecRmsd, match);

  bool overlay = true;
  OEChem.OERMSD(refMol, fitMol, vecRmsd, match, overlay);

  double[] rotmat = new double[9*nConfs];
  OEChem.OERMSD(refMol, fitMol, vecRmsd, match, overlay, rotmat);

  double[] transvec = new double[3*nConfs];
  OEChem.OERMSD(refMol, fitMol, vecRmsd, match, overlay, rotmat, transvec);
}
```

See also

- *OEWeightedRMSD* function
- *OEConfRMSD* class

Code Example

- *Align molecules by SMARTS match* example
- *Align molecules by clique match* example

OERotate

```
void OERotate(OEMolBase& mol, const float* m)
void OERotate(OEMolBase& mol, const double* m)
void OERotate(OEConfBase& conf, const float* m)
void OERotate(OEConfBase& conf, const double* m)
void OERotate(OEMCMolBase& mcmol, const float* m)
```

(continues on next page)

(continued from previous page)

```
void OERotate(OEMCMolBase& mcmol, const double* m)
```

These functions rotate a molecule by the 3×3 rotation matrix passed as the `m` argument that is 9-element array in row-major order. The rotation is defined as $xyz' = m \times xyz$, where m is the rotation matrix, xyz is the original vector, and xyz' is the new vector. The overloads for *OEConfBase* and *OEMCMolBase* are for efficiency.

mol
The molecule to rotate.

conf
The conformer to rotate.

mcmol
The multi-conformer molecule to rotate.

m
The rotation matrix.

See also

- Example program *MCS3DAlign.cs*

OESameAlternateLocation

```
bool OESameAlternateLocation(const OEAtomBase& res1,
                             const OEAtomBase& res2,
                             const bool allowBlank = false)
bool OESameAlternateLocation(const OEResidue& res1,
                             const OEResidue& res2,
                             const bool allowBlank = false)
```

Determines whether two atoms or their respective *OEResidue* s represent the same alternate location. This returns `true` when both residue have the same alternate location, or when one has a designated alternate location and the other has a blank (unspecified) alternate location. This does not enforce that the atoms to be in the same residue.

OESameChain

```
bool OESameChain(const OEAtomBase& atom1, const OEAtomBase& atom2)
bool OESameChain(const OEResidue& res1, const OEResidue& res2)
```

Determines whether two atoms or their respective *OEResidue* s are in the same chain. This returns `true` if both residues have the same NMR model number, *OEResidue.GetModelNumber*, and the same chain identifier, *OEResidue.GetChainID*.

OESameResidue

```
bool OESameResidue(const OEAtomBase& atom1, const OEAtomBase& atom2)
bool OESameResidue(const OEResidue& res1, const OEResidue& res2)
```

Determines whether atoms or their respective *OEResidue* s represent the same residue. This returns `true` when both residues are in the same chain (see *OESameChain*, have the same residue number, *OEResidue.GetResidueNumber*, the same sequence insertion code, *OEResidue.GetInsertCode*, and the same name, *OEResidue.GetName*.

OESameSubSearchScreenTypes

Attention

This API is currently available in C++ and Python.

```
bool OESameSubSearchScreenTypes(const OESubSearchScreen &,
                                const OESubSearchScreen &)
```

Returns whether or not the types (*OESubSearchScreenTypeBase*) of the two *OESubSearchScreen* objects are the same.

Warning

Screens with different types should never be compared with each other since they have been generated with different algorithms, they might have different lengths and their bits can represent different molecular features.

See also

- *OEGetSubSearchScreenType* function
- *OESubSearchScreenType* namespace

OEScrambleMolecule

```
bool OEScrambleMolecule(OEMolBase &)
```

Randomly reorders the atoms and bonds of a molecule. Like *OEMolBase.OrderAtoms* and *OEMolBase.OrderBonds*, this function modifies the order in which atoms and bonds are visited by the iterator returned by *OEMolBase.GetAtoms* and *OEMolBase.GetBonds* respectively. This function does not affect the atom or bond indices of any of the atoms or bonds of the molecule.

See also

- Example program *RandomizeAtoms.cs*

OEScreen

Attention

This API is currently available in C++ and Python.

```
bool OEScreen(const OESubSearchScreen& query,
              const OESubSearchScreen& target)
```

Returns whether an *OESubSearchScreen* object generated for a target molecule can be matched to the *OESubSearchScreen* object generated for a query molecule.

Example

> **Note**
>
> All screens provided by **OEChem TK** are rigorously tested to make sure that they do not eliminate any matches that would succeed when utilizing the more elaborate atom-by-atom substructure search algorithm.

> **See also**
>
> - *OEGetSubSearchScreenType* function
> - *OESubSearchScreenType* namespace

OESet2DHydrogenGeom

```
bool OESet2DHydrogenGeom(OEMolBase& mol)
```

Assigns approximate 2D coordinates to explicit hydrogen atoms of the given molecule.

> **See also**
>
> - *OESet2DHydrogenGeom* function
> - *OEAddExplicitHydrogens* function

OESet3DHydrogenGeom

```
bool OESet3DHydrogenGeom(OEMolBase& mol)
```

Assigns approximate 3D coordinates to explicit hydrogen atoms of the molecule. Only explicit hydrogen atoms with identical coordinates to their parent heavy atoms are updated.

```
bool OESet3DHydrogenGeom(OEMolBase& mol, const OEAtomBase* hatom)
```

Assigns approximate 3D coordinates to the given hydrogen.

> **See also**
>
> - *OESet2DHydrogenGeom* function
> - *OEAddExplicitHydrogens* function

OESetAngle

```
bool OESetAngle(OEMolBase& mol, OEAtomBase* a, OEAtomBase* b,
                OEAtomBase* c, double ang, double* axis = 0)
```

Sets the angle between atoms *a*, *b* and *c* to the value specified by the argument *ang* in radians. The three atoms must be bonded together in a molecule, with atom *b* being the vertex. All atoms which are attached to 'c' (excluding 'b') are rotated. If the bond defined by atoms 'b' and 'c' are in a ring, no change to the angle will occur. If an axis of rotation

is not provided, the rotation is performed such that the three atoms stay in the same plane. The function returns `true` if it completes successfully. The function returns `false` if the atoms are not bonded together, if the bond between *b* and *c* is part of a ring, or if the atoms are on top of each other.

mol
The molecule that the three atoms belong to.

a
One of the end atoms.

b
The vertex atom.

c
One of the end atoms.

ang
The angle in radians.

axis
The axis of rotation.

OESetAnisou

```
void OESetAnisou(OEAtomBase* atom, const OEAnisoUij& uij)
void OESetAnisou(OEAtomBase* atom, int u11, int u22, int u33, int u12,
                 int u13, int u23)
```

This function is used to set the anisotropic B-factor parameters associated with an *OEAtomBase*.

The form that uses a *OEAnisoUij* accesses the data packaged within the object.

See also

- *OEAnisoUij*
- *OEHasAnisou*
- *OEGetAnisou*

OESetAtomComment

```
void OESetAtomComment(OEAtomBase* atm, const char* c)
void OESetAtomComment(OEAtomBase* atm, const std::string& c)
```

Attaches a text comment to an individual atom. This function is used to manipulate atom alias information in MDL file formats. This property may be retrieved using the *OEGetAtomComment* function.

OESetCIPStereo

```
bool OESetCIPStereo(OEMolBase& mol, OEAtomBase* atom,
                    unsigned int cipstereo)
```

Sets the internal **OEChem TK** stereochemistry from the given CIP atom stereo descriptor. The descriptor must be either *OECIPAtomStereo.R* or *OECIPAtomStereo.S*. This function returns true if the stereochemistry was successfully set, and false otherwise: *i.e.* the descriptor was invalid or the specified atom was not a CIP atom stereo center.

See also

- *OEPerceiveCIPStereo* to perceive CIP atom stereo descriptor.
- *CIP Stereo Perception* section

```
bool OESetCIPStereo(OEMolBase& mol, OEBondBase* bond,
                    unsigned int cipstereo)
```

Sets the internal **OEChem TK** stereochemistry from the given CIP atom stereo descriptor. The descriptor must be either *OECIPBondStereo.E* or *OECIPBondStereo.Z*. This function returns true if the stereochemistry was successfully set, and false otherwise: *i.e.* the descriptor was invalid or the specified bond was not a CIP bond stereo center.

See also

- *OEPerceiveCIPStereo* to perceive CIP bond stereo descriptor.
- *CIP Stereo Perception* section

OESetComment

```
void OESetComment(OEMolBase& mol, const char* c)
void OESetComment(OEMolBase& mol, const std::string& c)
```

Attaches a text comment to a molecule. This comment will be written to the comment field of the .mol2 and .sdf molecular formats. It will also be conserved in OEBinary format. This property may be retrieved using the *OEGetComment* function.

OESetDefaultFlavors

```
bool OESetDefaultFlavors(oemolistream &)
```

Sets the flavors for all of the file formats associated with the *oemolistream* object to their default state. The *oemolistream* constructor invokes this function to initialize its input file format flavors.

```
bool OESetDefaultFlavors(oemolostream &)
```

Sets the flavors for all of the file formats associated with the *oemolostream* object to their default state. The *oemolostream* constructor invokes this function to initialize its output file format flavors.

See also

- *OESetDefaultIFlavors* and *OESetDefaultOFlavors* functions
- *OEGetDefaultIFlavor* and *OEGetDefaultOFlavor* functions
- *the OEFormat namespace*
- *OEIFlavor* and *OEOFlavor* namespaces
- *oemolstreambase.SetFlavor* and *oemolstreambase.GetFlavor* methods

OESetDefaultIFlavors

```
bool OESetDefaultIFlavors(oemolstreambase &)
```

Sets the flavors for all of the formats associated with the *oemolstreambase* object to their default state. The default flavors are automatically set by the *oemolistream* constructors.

See also

- *OEGetDefaultIFlavor* function
- *OEIFlavor* and *the OEFormat namespace*
- *oemolstreambase.SetFlavor* and *oemolstreambase.GetFlavor* methods

OESetDefaultOFlavors

```
bool OESetDefaultOFlavors(oemolstreambase &)
```

Sets the flavors for all of the formats associated with the *oemolstreambase* object to their default state. The default flavors are automatically set by the *oemolostream* constructors.

See also

- *OEGetDefaultOFlavor* function
- *OEOFlavor* and *the OEFormat namespace*
- *oemolstreambase.SetFlavor* and *oemolstreambase.GetFlavor* methods

OESetDimensionFromCoords

```
void OESetDimensionFromCoords(OEMolBase &)
```

Assigns the molecule's dimension property from the coordinates using the *OEMolBase.SetDimension* method. The dimension property is set to the value `0`, `1`, `2` or `3` depending upon the number of ordinates for which any atom has a non-zero value.

The table below illustrates the difference between the *OESetDimensionFromCoords* and *OESetMDLDimensionFromCoords* functions.

Coordinates			OESetDimensionFromCoords	OESetMDLDimensionFromCoords
X	Y	Z		
0.00	0.00	0.00	0D	0D
1.00	0.00	0.00	1D	2D
0.00	1.00	0.00	1D	2D
0.00	0.00	1.00	1D	3D
1.00	1.00	0.00	2D	2D
0.00	1.00	1.00	2D	3D
1.00	0.00	1.00	2D	3D
1.00	1.00	1.00	3D	3D

See also

- *OEGetDimensionFromCoords* function
- *OEGetMDLDimensionFromCoords* and *OESetMDLDimensionFromCoords* functions

OESetDistance

```
bool OESetDistance(OEMolBase& mol, OEAtomBase* a, OEAtomBase* b,
                   double dist)
```

Sets the distance between two atoms to the value specified by the argument *dist*. The two atoms must be bonded together. The atom passed in as *b* is displaced. If the bond defined by atoms is in a ring, no change to the distance will occur. The function returns `true` if it completes successfully. The function returns `false` if the atoms do not form a bond, if the bond is part of a ring, or if the atoms are on top of each other.

mol
The molecule that the pair of atom belongs to.

a
The first atom.

b
The second atom.

dist
The distance.

OESetMCSExhaustiveSearchTruncationLimit

```
void OESetMCSExhaustiveSearchTruncationLimit(unsigned int limit)
```

Sets the number of unique partial substructures that can be traversed during the *exhaustive* MCS search. If this limit is reached than the *exhaustive* MCS search throws a "MCS search truncated" waning and terminates returning the maximum common substructures identified prior to reaching this limit.

Warning

Use this function with caution. Reducing the default limit will result in terminating the exhaustive MCS searches prematurely thereby preventing them from locating the maximum common substructure(s). Increasing this limit can result in significantly increasing the execution time in case of complex or large molecule pairs.

See also

- *OEGetMCSExhaustiveSearchTruncationLimit* function
- *OEMCSSearch* class.
- *Maximum Common Substructure Search* section

OESetMDLDimensionFromCoords

```
void OESetMDLDimensionFromCoords(OEMolBase &)
```

Assigns the molecule's dimension property from the coordinates using the *OEMolBase.SetDimension* method. The dimension property is set to the value:

- 0 if all x, y, z coordinates are zero
- 2 if there is at least one atom with non-zero x or z coordinates
- 3 it there is at least one atom with non-zero z coordinate

The table below illustrates the difference between the *OESetDimensionFromCoords* and *OESetMDLDimensionFromCoords* functions.

Coordinates			OESetDimensionFromCoords	OESetMDLDimensionFromCoords
X	Y	Z		
0.00	0.00	0.00	0D	0D
1.00	0.00	0.00	1D	2D
0.00	1.00	0.00	1D	2D
0.00	0.00	1.00	1D	3D
1.00	1.00	0.00	2D	2D
0.00	1.00	1.00	2D	3D
1.00	0.00	1.00	2D	3D
1.00	1.00	1.00	3D	3D

See also

- *OEGetMDLDimensionFromCoords* function
- *OEGetDimensionFromCoords* and *OESetDimensionFromCoords* functions

OESetMMCIFChemCompData

```
bool OESetMMCIFChemCompData(OEMolBase& mol,
                            const OEMolBase& chemCompMol,
                            const bool strict = false)
```

Updates _chem_comp sections in the MMCIF header metadata with information from the input chemCompMol molecule. First, a single-residue chemCompMol is checked to name match with a residue in mol molecule. The chemCompMol atom and bond data is then used to replace the _chem_comp header data for a similarly named residue in the mol. If strict is used, an additional check for atom name matching is made between the matched residue in the molecule and the chemCompMol.

The following categories need to be present and properly populated in the input molecules header for this function. If any attributes are missing, running OEUpdateMMCIFData with ChemComp perception should populate any missing attributes:

- _chem_comp.
- _chem_comp_atom.
- _chem_comp_bond.

If the molecule already exists, it will replace all chemical component references of that molecule with the new molecule. By default, residue atom name matching is enforced. If strict is turned off, an *OESubSearch* is used to identify structural similarity with the new molecule. If a match is found, the matching structure atom names will be used in the _chem_comp atom naming scheme.

See also

- *OECIFData* class for cif data manipulation
- *mmCIF Metadata Manipulation* section

Code Example

- *Setting mmCIF Chemical Component Header* example

OESetMMCIFData

```
bool OESetMMCIFData(OEMolBase& mol, const std::string& value)
bool OESetMMCIFData(OEMolBase& mol, const OECIFData& cifData)
```

Sets mmCIF metadata on a molecule. If metadata already exists, it is replaced with the new value.

See also

- *OEAddMMCIFData* function to add MMCIF data
- *mmCIF Metadata Manipulation* section

Note

Unicode characters are not supported in either the data tag or the data value.

OESetOEGraphMol

```
boolean OESetOEGraphMol(OEBase obj, String tag, OEGraphMol mol)
```

Attaches a copy of the *OEGraphMol* `mol` to the *OEBase* `obj` using the `tag` as *generic data*.

See also

- The *Attaching other objects* section for example code
- *OEGetOEGraphMol*

OESetOEMol

```
boolean OESetOEMol(OEBase obj, String tag, OEMol mol)
```

Attaches a copy of the *OEMol* `mol` to the *OEBase* `obj` using the `tag` as *generic data*.

See also

- The *Attaching other objects* section for example code
- *OEGetOEMol*

OESetOEQMol

```
boolean OESetOEQMol(OEBase obj, String tag, OEQMol mol)
```

Attaches a copy of the *OEQMol* `mol` to the *OEBase* `obj` using the `tag` as *generic data*.

See also

- The *Attaching other objects* section for example code
- *OEGetOEQMol*

OESetPackedCoords

```
void OESetPackedCoords(OEMolBase& mol, const float* coords)
void OESetPackedCoords(OEMolBase& mol, const double* coords)
```

Sets the coordinates of the molecule to the coordinates passed in the coords argument. The coords array must be at least of size *OEMolBase.NumAtoms* *3. The atom coordinates in the coords array should be packed and they should be in the same order as the atoms in the iterator returned by *OEMolBase.GetAtoms*.

mol
: The molecule to set coordinates.

coords
: The array with coordinates.

OESetPDBData

```
bool OESetPDBData(OEMolBase& mol, const OESDDataPair& dp)
bool OESetPDBData(OEMolBase& mol, const std::string& tag,
                  const std::string& value)
```

Sets a PDB tag and value data pair to a molecule. Both the tag and the value must be a string. If an item with the same tag already exists, it is replaced with the new value.

See also

- *OEAddPDBData* function to add multiple PDB data
- *PDB Tagged Data Manipulation* section

Note

Unicode characters are not supported neither in the data tag nor in the data value.

OESetSDData

```
bool OESetSDData(OEMolBase& mol, const OESDDataPair& dp)
bool OESetSDData(OEMolBase& mol, const std::string& tag,
                 const std::string& value)
```

Sets an SD tag and value data pair to a molecule. Both the tag and the value must be a string. If an item with the same tag already exists, it is replaced with the new value.

See also

- *OEAddSDData* function to add multiple SD data with the same tag
- *OEGetSDData* function
- *OEGetSDDataPairs* function to access multiple items with the same tag
- *SD Tagged Data Manipulation* section

Code Example

- *Adding csv data as SD tags* example

Note

Unicode characters are not supported neither in the data tag nor in the data value.

OESetStyle

```
void OESetStyle(OEChem::OEMolBase& mol, const OE3DMolStyle& style)
void OESetStyle(OEChem::OEAtomBase& atom, const OE3DMolStyle& style)
void OESetStyle(OESystem::OEIter<OEChem::OEAtomBase>& atoms,
                const OE3DMolStyle& style)
```

Sets the *OE3DMolStyle* of a *OEMolBase*, *OEAtomBase*, or an iterator of *OEAtomBase*.

OESetStyleIndex

```
void OESetStyleIndex(OEAtomBase& atom, unsigned int idx)
void OESetStyleIndex(OEConfBase& conf, unsigned int idx)
```

Sets the style index of a specified *OEAtomBase* or *OEConfBase*.

OESetTorsion

```
bool OESetTorsion(OEMolBase& mol, OEAtomBase* a, OEAtomBase* b,
                  OEAtomBase* c, OEAtomBase* d, double ang)
bool OESetTorsion(OEConfBase& conf, OEAtomBase* a, OEAtomBase* b,
                  OEAtomBase* c, OEAtomBase* d, double ang)
```

Sets torsion angle defined by atoms *a*, *b*, *c* and *d* to the angle in the argument radians. It is presumed that each of these atoms are members of the molecules passed to the function. All atoms which are attached to 'c' (excluding 'b') are rotated. If the bond defined by atoms 'b' and 'c' are in a ring, no change to the torsion will occur. However, an overall rotation of the molecule may be a side effect. The functions return `true` if completes successfully. The functions return `false` if the central bond is part of a ring, or if the atoms are on top of each other.

mol
: The molecule that the atoms belong to.

conf
: The conformer that the atoms belong to.

a
: One of the end atoms in the torsion.

b
: Atom that is connected to *a* in the torsion.

c
: Atom that is connected to *d* in the torsion.

d
: The other end atom in the torsion.

ang
: The angle in radians.

OEShortestPath

Table 41: **The overloaded versions of the OEShortestPath function**

Link	Description
OEShortestPath(atomA, atomB)	find shortest path between two atoms
OEShortestPath(atomA, atomB, atompred)	find shortest path between two atoms (excluding certain atoms)
OEShortestPath(atomA, atomB, bondpred)	find shortest path between two atoms (excluding certain bonds)

```
OESystem::OEIterBase<OEAtomBase>* OEShortestPath(const OEAtomBase* a,
                                                 const OEAtomBase* b)
```

Returns a shortest path between the two given atoms. If no path exists the *OEShortestPath* function returns an empty iterator.

Example

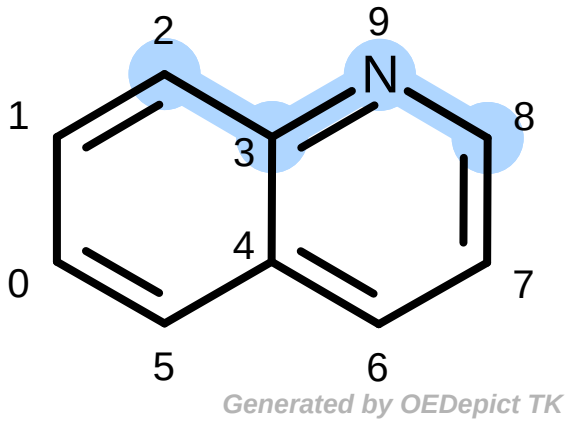


Fig. 14: **Example of shortest path between atom 2 and 8**

Note

The shortest path returned by the *OEShortestPath* function is non-unique. There are more than one valid shortest path can exist between two atoms. The *OEShortestPath* only returns one of them. For example, there are three valid six atoms long shortest path exist between atom 1 and 7 in the example above: `1, 2, 3, 9, 8, 7`, `1, 0, 5, 4, 6, 7`, and `1, 2, 3, 4, 6, 7`.

```
OESystem::OEIterBase<OEAtomBase>* OEShortestPath(const OEAtomBase* a,
                                                 const OEAtomBase* b,
                                                 const OESystem::OEUnaryPredicate
↪<OEAtomBase>& excludeatoms)
```

Returns a shortest path between the two given atoms. The returned path will exclude any atom that satisfies the given atom predicate. If no path exists the *OEShortestPath* function returns an empty iterator.

Example

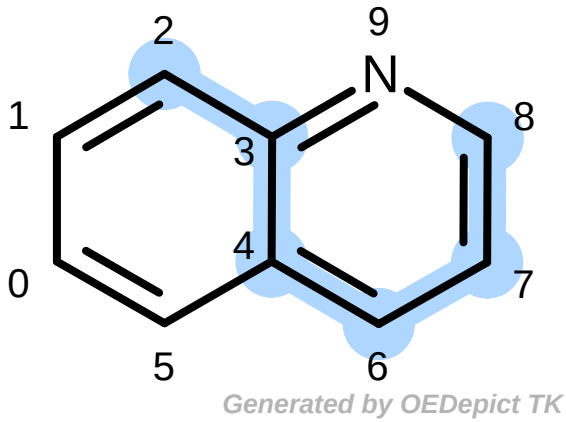


Fig. 15: **Example of shortest path between atom 2 and 8 without traversing any nitrogen atom**

```
OESystem::OEIterBase<OEAtomBase>* OEShortestPath(const OEAtomBase* a,
                                                 const OEAtomBase* b,
                                                 const OESystem::OEUnaryPredicate
↪<OEBondBase>& excludebonds)
```

Returns a shortest path between the two given atoms. The returned path will exclude any bond that satisfies the given bond predicate. If no path exists, the *OEShortestPath* function returns an empty iterator.

See also

- *OEGetPathLength* function
- Example program *MinPath.cs*

OESingleBondCount

```
unsigned int OESingleBondCount(const OEAtomBase* atm)
```

Returns the number of single bonds connected to an atom. The return value includes the number of implicit hydrogens attached to an atom, as returned by the *OEAtomBase.GetImplicitHCount* method.

OESmartsLexReplace

Replaces all the instances of `$varname` in the SMARTS pattern `smarts` with the variables as defined by the mapping in `bindings`.

OESmilesAtomCount

```
unsigned int OESmilesAtomCount(const char* str)
unsigned int OESmilesAtomCount(const std::string& str)
```

This function efficiently determines the number of **heavy** atoms in a SMILES string, without converting it into an *OEMolBase*.

OECXSMILESToMol

```
bool OECXSMILESToMol(OEMolBase& mol, const std::string& str)
```

Converts a CXSMILES string into a molecule. After constructing the molecule the following properties are perceived:

- Rings are perceived by calling the *OEFindRingAtomsAndBonds* function.
- The aromaticity is assigned by calling the *OEAssignAromaticFlags* function using the *OEAroModel.OpenEye* aromaticity model.
- The chirality is perceived by calling the *OEPerceiveChiral* function.

Note that since CXSMILES is a superset of the SMILES syntax, it is safe to use this function for either format. Using SMILES readers for known CXSMILES is discouraged since any appendix information will become a static addition to the molecule title and cannot be recovered.

The molecule object is cleared before parsing so the resulting molecule object only contains atoms and bonds from the CXSMILES. If one wishes to *add* atoms and bonds to an existing molecule, one can use the lower-level *OEParseSmiles* function (with an option argument that enables CXSMILES parsing), which preserves any existing atoms and bonds in the molecule.

See also

- *OEMolToCXSMILES* function
- *OEMolToSmiles* function
- *Unique Representation* section

OESmilesToMol

```
bool OESmilesToMol(OEMolBase& mol, const std::string& str)
```

Converts a SMILES string into a molecule. After constructing the molecule the following properties are perceived:

- Rings are perceived by calling the *OEFindRingAtomsAndBonds* function.
- The aromaticity is assigned by calling the *OEAssignAromaticFlags* function using the *OEAroModel.OpenEye* aromaticity model.
- The chirality is perceived by calling the *OEPerceiveChiral* function.

The molecule object is cleared before parsing so the resulting molecule object only contains atoms and bonds from the SMILES. If one wishes to *add* atoms and bonds to an existing molecule, one can use the lower-level *OEParseSmiles* function, which preserves any existing atoms and bonds in the molecule.

See also

- *OEMolToSmiles* function
- *OEMolToCXSMILES* function
- *Unique Representation* section

OESortConfsBySDTag

```
bool OESortConfsBySDTag(OEChem::OEMCMolBase& mol,
                        const std::string& sdtag, bool biggerIsBetter)
```

Reorders the conformers of *mol* based on SD values associated with the given *sdtag*. The SD values must be numeric. *biggerIsBetter* controls if the highest (true) values appear first in the list or the lowest values (false).

Conformers missing *sdtag* SD data will be sorted to the end of the list.

OESortConfsByTag

```
bool OESortConfsByTag(OEChem::OEMCMolBase& mol,
                      const std::string& tag, bool biggerIsBetter)
```

Reorders the conformers of *mol* based on the value of generic data on the conformers with the specified *tag*. The generic data must be stored as either a float or double. *biggerIsBetter* controls if the highest (true) values appear first in the list or the lowest values (false).

Conformers missing *tag* float/double generic data will best sorted to the end of the conformer list.

OEString2OEAtomExpr

```
unsigned int OEString2OEAtomExpr(const char* expression)
```

OEString2OEBondExpr

```
unsigned int OEString2OEBondExpr(const char* expression)
```

OESubSearchStatusToName

Attention

This API is currently available in C++ and Python.

```
std::string OESubSearchStatusToName(const unsigned status)
```

Returns the string representation of the constants in the *OESubSearchStatus* namespace.

See also

- *OESubSearchDatabase.Search* method

Code Example

- *Perform SMARTS search on substructure database file* example

The following functions generate fully-functional molecule from a subset of another molecule. The following subset methods are available:

- *Match-based selection* methods

- *Atom predicate-based selection* methods
- *Atom and bond predicate-based selection* methods
- *Atom and bond set-based selection* methods

OESubsetMol

Match-based selection

```
bool OESubsetMol(OEMolBase& dst, const OEMatchBase* match,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEMCMolBase& dst, const OEMCMolBase& src,
                 const OEMatchBase* match, bool adjustHCount = false,
                 bool RGroup = false, OEAtomBase** atomMap = 0,
                 OEBondBase** bondMap = 0)
```

dst
The destination subset molecule generated from the source molecule. The source molecule may be multi-conformer, in which case a multi-conformational subset molecule is produced.

src
The source molecule.

match
The container that stores the selected atoms that will be in the generated subset molecule. By default, bonds between the selected atoms are carried along. If no source molecule is specified then the target molecule of the *OEMatchBase* defines the source molecule.

RGroup
Flag that indicates whether "R groups" are to be added when covalent bonds are broken by the selection.

adjustHCount
Flag that indicates whether the implicit hydrogen count is to be adjusted when covalent bonds are broken by the selection in order to preserve the original hybridization.

atomMap, bondMap
A mapping between the atoms or / and bonds of the source and destination molecule can be obtained by passing in the `atomMap` and `bondMap` arrays respectively. These arrays should be large enough to hold all the atoms / bonds for the source molecule, *i.e.*, the size of *OEMolBase.GetMaxAtomIdx* and *OEMolBase.GetMaxBondIdx*, respectively. The atoms or bonds in the array are those from the destination molecule placed at the source molecule's atom / bond index location in the array.

Atom predicate-based selection

```
bool OESubsetMol(OEMolBase& dst, const OEMolBase& src,
                 const OESystem::OEUnaryPredicate<OEAtomBase>& atomfcn,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEMCMolBase& dst, const OEMCMolBase& src,
                 const OESystem::OEUnaryPredicate<OEAtomBase>& atomfcn,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEQMolBase& dst, const OEQMolBase& src,
                 const OESystem::OEUnaryPredicate<OEAtomBase>& atomfcn,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
```

dst
The destination subset molecule generated from the source molecule. The source molecule may be multi-conformer, in which case a multi-conformational subset molecule is produced.

src
The source molecule.

atomfcn
The atom predicate that defines the subset of atoms that are carried over to the destination molecule. By default, bonds between the selected atoms are carried along.

RGroup
Flag that indicates whether "R groups" are to be added when covalent bonds are broken by the selection.

adjustHCount
Flag that indicates whether the implicit hydrogen count is to be adjusted when covalent bonds are broken by the selection in order to preserve the original hybridization.

atomMap, bondMap
A mapping between the atoms or / and bonds of the source and destination molecule can be obtained by passing in the `atomMap` and `bondMap` arrays respectively. These arrays should be large enough to hold all the atoms / bonds for the source molecule, *i.e.*, the size of *OEMolBase.GetMaxAtomIdx* and *OEMolBase.GetMaxBondIdx*, respectively. The atoms or bonds in the array are those from the destination molecule placed at the source molecule's atom / bond index location in the array.

Atom and bond predicate-based selection

```
bool OESubsetMol(OEMolBase& dst, const OEMolBase& src,
                 const OESystem::OEUnaryPredicate<OEAtomBase>& atomfcn,
                 const OESystem::OEUnaryPredicate<OEBondBase>& bondfcn,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEMCMolBase& dst, const OEMCMolBase& src,
                 const OESystem::OEUnaryPredicate<OEAtomBase>& atomfcn,
                 const OESystem::OEUnaryPredicate<OEBondBase>& bondfcn,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEQMolBase& dst, const OEQMolBase& src,
                 const OESystem::OEUnaryPredicate<OEAtomBase>& atomfcn,
                 const OESystem::OEUnaryPredicate<OEBondBase>& bondfcn,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
```

dst
The destination subset molecule generated from the source molecule. The source molecule may be multi-conformer, in which case a multi-conformational subset molecule is produced.

src
The source molecule.

atomfcn
The atom predicate that defines the subset of atoms that are carried over to the destination molecule.

bondfcn
The atom predicate that defines the subset of bonds that are carried over to the destination molecule.

RGroup
Flag that indicates whether "R groups" are to be added when covalent bonds are broken by the selection.

adjustHCount
Flag that indicates whether the implicit hydrogen count is to be adjusted when covalent bonds are broken by the selection in order to preserve the original hybridization.

atomMap, bondMap
A mapping between the atoms or / and bonds of the source and destination molecule can be obtained by passing in the `atomMap` and `bondMap` arrays respectively. These arrays should be large enough to hold all the atoms / bonds for the source molecule, *i.e.*, the size of *OEMolBase.GetMaxAtomIdx* and *OEMolBase.GetMaxBondIdx*, respectively. The atoms or bonds in the array are those from the destination molecule placed at the source molecule's atom / bond index location in the array.

Atom and bond set-based selection

```
bool OESubsetMol(OEMolBase& dst, const OEAtomBondSet& abset,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEMolBase& dst, const OEMolBase& src,
                 const OEAtomBondSet& abset,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEMCMolBase& dst, const OEMCMolBase& src,
                 const OEAtomBondSet& abset,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
bool OESubsetMol(OEQMolBase& dst, const OEQMolBase& src,
                 const OEAtomBondSet& abset,
                 bool adjustHCount = false, bool RGroup = false,
                 OEAtomBase** atomMap = 0, OEBondBase** bondMap = 0)
```

dst
The destination subset molecule generated from the source molecule. The source molecule may be multi-conformer, in which case a multi-conformational subset molecule is produced.

src
The source molecule.

abset
The container that stores the selected atoms and bonds that will be in the generated subset molecule. If no source molecule is specified then *OEAtomBondSet* defines the source molecule.

RGroup
Flag that indicates whether "R groups" are to be added when covalent bonds are broken by the selection.

adjustHCount
Flag that indicates whether the implicit hydrogen count is to be adjusted when covalent bonds are broken by the selection in order to preserve the original hybridization.

atomMap, bondMap
A mapping between the atoms or / and bonds of the source and destination molecule can be obtained by passing in the `atomMap` and `bondMap` arrays respectively. These arrays should be large enough to hold all the atoms / bonds for the source molecule, *i.e.*, the size of *OEMolBase.GetMaxAtomIdx* and *OEMolBase.GetMaxBondIdx*, respectively. The atoms or bonds in the array are those from the destination molecule placed at the source molecule's atom / bond index location in the array.

See also

- *Parallel Data Structures* section
- *Molecule Partitioning* section
- Example program *Parts2Mols.cs*
- Example program *ExtractScaffold.cs*

OESuppressHydrogens

```
bool OESuppressHydrogens(OEMolBase& mol, bool retainPolar = false,
                         bool retainStereo = false,
                         bool retainIsotope = true)
bool OESuppressHydrogens(OEAtomBase* atom, bool retainPolar = false,
                         bool retainStereo = false,
                         bool retainIsotope = true)
```

Transforms the explicit hydrogens in a molecule into implicit hydrogens on their parent heavy atom. The overloaded function provides the same capability at a specific heavy atom.

Any explicit hydrogen atoms, those with atomic number *OEElemNo.H*, are deleted and any information associated with them, coordinates, isotopes, generic data etc... are lost. In their place, the implicit hydrogen count field of the heavy atom to which they are bonded is appropriately incremented.

retainPolar
: If `true`, then hydrogens on polar atoms, *i.e.* those for which *OEAtomBase.IsPolarHydrogen* returns `true`, are left unaffected.

retainStereo
: If `true`, then hydrogens required to specify stereo centers, *i.e.* those for which *OEHasStereoHydrogens* return `true`, are left unaffected.

retainIsotope
: If `true`, then specified isotopes of hydrogen (including deuterium `[2H]`, tritium `[3H]` and protium `[1H]`) will be treated as heavy atoms and retained.

The functions return the value `true` if any hydrogens were deleted, and return `false` if the molecule was left unchanged.

See also

- *OEAddExplicitHydrogens* function
- *Align molecules by clique match* **OEChem TK** example program

OESweepMDLStereoGroups

```
bool OESweepMDLStereoGroups(OEMolBase& mol,
                            bool clearmdlstereomarks = true)
```

Removes all atoms that lack atom stereo parity information from enhanced stereogroups. If the resulting stereogroup basis is empty, the group itself is also deleted. The `clearmdlstereomarks` argument controls whether stereomarks to the identified nonstereo atom(s) should also be removed. Stereomarks to atoms that are not a member of any enhanced stereogroup are unaffected.

A return status of `true` indicates that the provided molecule has been altered.

See also

- *OEIsMDLStereoGroup*
- *GetGroups*
- *OEBondStereo*

OESweepRotorCompressHydrogens

```
bool OESweepRotorCompressHydrogens(OEMCMolBase& mol,
                                   bool retainPolar = false,
                                   bool retainStereo = false,
                                   bool retainIsotope = true)
```

Suppress hydrogens from a rotor-offset-compressed *OEMCMolBase* in a way that does not corrupt the rotor compression data tables. Removing explicit hydrogens can have a dramatic improvement on the storage requirements of a molecule as hydrogen coordinates do not need to be stored anymore.

The `retainPolar`, `retainStereo`, `retainIsotope` arguments are identical to the arguments passed to the *OESuppressHydrogens* function and are forwarded to that function.

Return `false` if an error like the following is detected: the molecule contains deleted atoms; or the rotor compression data is corrupted in some way. Return `true` if sweeping was successful and *OEMolBase.Sweep* was called on the molecule. Also return `true` if there were no hydrogens to sweep away at all and nothing was done.

Warning

The returned boolean value has a different meaning than *OESuppressHydrogens* boolean return value. *OESuppressHydrogens* will return `false` whenever a molecule does not contain hydrogens.

OESweepStyles

```
void OESweepStyles(OEMolBase& mol)
```

Utilized when merging two *OE3DMolStyle* to make sure the styles are appropriatly set to the atoms and the molecule.

OESymmetryNumber

```
unsigned OESymmetryNumber(const OEMolBase& mol, bool useH = true,
                          double threshold = 0.05)
bool OESymmetryNumber(const OEMCMolBase& mol, unsigned* symCtArray,
                      bool useH = true, double threshold = 0.05)
```

Calculates the symmetry number of the molecule. For the overload with *OEMolBase* as input molecule, the symmetry number is the return value. For the *OEMCMolBase* overload, the symmetry numbers are returned in the *symCtdArray* array. The *symCtArray* passed to this function should be of length `fit.GetMaxConfIdx()`. The *useH* flag indicates whether symmetry calculation should use the hydrogen atoms of the molecule or ignore them. Root mean squared deviation between conformations are performed internally and accepted as identical based on the specified *threshold*. For the *OEMCMolBase* overload the return value is `True` if calculation completes successfully, and `False` otherwise.

mol
The molecule to calculate symmetry number.

useH
Flag indicating if hydrogen atoms should be considered explicitly.

threshold
Threshold for root mean squared deviation (RMSD) comparison.

symCtArray
Array containing calculated symmetry numbers in a multi-conformer input molecule.

OETerminalOCount

```
unsigned int OETerminalOCount(const OEAtomBase* atm)
```

OETerminalSCount

```
unsigned int OETerminalSCount(const OEAtomBase* atm)
```

OETheFunctionFormerlyKnownAsStripSalts

```
bool OETheFunctionFormerlyKnownAsStripSalts(OEMolBase& mol)
```

Deletes all atoms that are not part of the largest connected component of a molecule. If two or more components of a molecule are the size of the largest component, this function keeps the first one encountered.

Note

The *OETheFunctionFormerlyKnownAsStripSalts* function is identical to *OEDeleteEverythingExceptTheFirstLargestComponent*.

See also

- Example program *StripSalts.cs*

OETranslate

```
void OETranslate(OEMolBase& mol, const float* v)
void OETranslate(OEMolBase& mol, const double* v)
void OETranslate(OEConfBase& conf, const float* v)
void OETranslate(OEConfBase& conf, const double* v)
void OETranslate(OEMCMolBase& mcmol, const float* v)
void OETranslate(OEMCMolBase& mcmol, const double* v)
```

Translates the molecule by the vector passed as the argument 'v'. The vector 'v' must be of size three and contain the Cartesian translation vectors. The overloads for *OEConfBase* and *OEMCMolBase* are for efficiency.

mol
The molecule to translate.

conf
The conformer to translate.

mcmol
The multi-conformer molecule to translate.

v
The translation vector.

See also

- Example program *MCS3DAlign.cs*

OETrimConformers

```
bool OETrimConformers(OEChem::OEMCMolBase& mol, unsigned int N)
```

Deletes conformers of *mol*, starting with those with the highest conformer index, such *mol* has *N* conformers remaining.

If *mol* has less than *N* conformers, or *N* is passed 0 then this function is a no-op.

OETripleBondCount

```
unsigned int OETripleBondCount(const OEAtomBase* atm)
```

Returns the number of triple bonds connected to an atom.

OETriposAtomNames

```
void OETriposAtomNames(OEMolBase& mol)
```

Sets the atom name property of each atom, using the *OEAtomBase.SetName* method, to follow the convention typically followed in Sybyl .mol2 files. Each atom is named by its atomic symbol (or `Du` for element zero) followed by a sequential index per atomic number. For example, the first carbon in the molecule is named `C1`, the second is named `C2`, and so on. Similarly, the first oxygen is named `O1` and so on. The 'atom name' property may be retrieved using the *OEAtomBase.GetName* method.

The ordering of atoms is taken from the iterator returned by the *OEMolBase.GetAtoms* method (and may therefore be changed by using *OEMolBase.OrderAtoms*) and is unchanged by this function.

See also

- Example program *PrintAtomNames.cs*
- *OETriposAtomTypes*
- *OETriposAtomTypeNames*

OETriposAtomTypeNames

```
bool OETriposAtomTypeNames(OEMolBase& mol)
```

Sets the string atom type property of every atom in a molecule to its Tripos Sybyl atom type. The symbolic atom type of each atom is determined by *OETriposAtomType* and then is converted from integer index to type name using *OETriposTypeName*. The string 'atom type property' is set using the *OEAtomBase.SetType* method. The 'atom type name' property may be retrieved using the *OEAtomBase.GetType* method. This function returns `false` if any atom was assigned with `"Du"`, and returns `true` otherwise.

This function prefers that aromaticity has previously been assigned using the Tripos model of aromaticity, see the *OEAssignAromaticFlags* function. However, reasonable atom types are returned when using other models of aromaticity.

See also

- *OETriposAtomTypes*
- *OETriposTypeNames*
- *OETriposAtomNames*

OETriposAtomType

```
unsigned int OETriposAtomType(const OEAtomBase* atm)
```

This function determines the Tripos Sybyl atom type for the given atom. The result is an unsigned integer index taken from the *OETriposType* namespace. The index zero corresponds to *OETriposType.Du*, *i.e.* a dummy or unrecognized atom. The atom, including any previously assigned integer atom type, is not modified by this function.

This function prefers that aromaticity has previously been assigned using the Tripos model of aromaticity, see the *OEAssignAromaticFlags* function. However, reasonable atom types are returned when using other models of aromaticity.

OETriposAtomTypes

```
bool OETriposAtomTypes(OEMolBase& mol)
```

This function sets the integer atom type property of every atom in a molecule to its Tripos Sybyl atom type. The integer atom types are taken from the *OETriposType* namespace, as determined by the *OETriposAtomType* function, and each atom is assigned a value using the *OEAtomBase.SetIntType* method. The *OETriposAtomTypes* function returns `false` if any atom was assigned *OETriposType.Du*, and returns the value `true` otherwise.

This function prefers that aromaticity has previously been assigned using the Tripos model of aromaticity, see the *OEAssignAromaticFlags* function. However, reasonable atom types are returned when using other models of aromaticity.

OETriposBondTypeNames

```
void OETriposBondTypeNames(OEMolBase& mol)
```

Sets the string bond type property of every bond in a molecule to its Tripos Sybyl bond type. Single, double and triple bonds are assigned the types `1`, `2` and `3` respectively with the exception of aromatic and amide bonds with are assigned the values `ar` and `am` respectively.

This function prefers that aromaticity has previously been assigned using the Tripos model of aromaticity, see the *OEAssignAromaticFlags* function. However, reasonable atom types are returned when using other models of aromaticity.

OETriposOrderAtoms

```
void OETriposOrderAtoms(OEMolBase& mol)
```

OETriposResonantTypeNames

```
void OETriposResonantTypeNames(OEMolBase& mol)
```

OETriposTypeElement

```
unsigned int OETriposTypeElement(unsigned int type)
```

Returns the atomic number/element represented by the given Tripos atom type index, as encoded by the *OETriposType* namespace.

OETriposTypeIndex

```
unsigned int OETriposTypeIndex(const char* name)
```

Returns the integer atom type, as encoded by the *OETriposType* namespace, that corresponds to the given Tripos Sybyl atom type string. For example, *OETriposTypeIndex* returns the integer value *OETriposType.Npl3* for "N.pl3".

This function is the inverse of *OETriposTypeName*.

OETriposTypeName

```
const char* OETriposTypeName(unsigned int type)
```

Returns the symbolic atom type name that corresponds to the given Tripos Sybyl atom type index, as encoded by the *OETriposType* namespace. For example, *OETriposTypeName* returns the value "N.pl3" for *OETriposType.Npl3*.

This function is the inverse of *OETriposTypeIndex*.

OETriposTypeNames

```
bool OETriposTypeNames(OEMolBase& mol)
```

Loops through the atoms of the molecule, setting the string atom type property of each atom, to atom type name of the Tripos Sybyl atom type encoded in the atoms integer atom type property. This is approximately equivalent to the following operation:

```
atm->SetType(OETriposTypeName(atm->GetIntType()))
```

The 'atom type name' property may be retrieved using the *OEAtomBase.GetType* method. If both the string and integer atom types are required, it is faster to call *OETriposAtomTypes* and then *OETriposTypeNames* (which reuses the results of the first call) than it is to call *OETriposAtomTypes* and *OETriposAtomTypeNames*.

See also

- *OETriposAtomNames*

OEUncolorMol

```
bool OEUncolorMol(OEChem::OEMolBase& uncoloredcmol,
                  const OEChem::OEMolBase& inputmol,
                  unsigned uncolortype = OEUncolorStrategy::Default)
bool OEUncolorMol(OEChem::OEMCMolBase& uncoloredcmol,
```

(continues on next page)

(continued from previous page)

```
                  const OEChem::OEMCMolBase& inputmol,
                  unsigned uncolortype = OEUncolorStrategy::Default)
bool OEUncolorMol(OEChem::OEMolBase& moltouncolor,
                  unsigned uncolortype = OEUncolorStrategy::Default)
bool OEUncolorMol(OEChem::OEMCMolBase& moltouncolor,
                  unsigned uncolortype = OEUncolorStrategy::Default)
```

The *OEUncolorMol* function will remove chemical features from the output structure according to the *OEUncolorStrategy*. The default strategy is *OEUncolorStrategy.RemoveDimension* and *OEUncolorStrategy.BemisMurcko*.

For the function overloads that operate directly on the input molecule, the function returns `true` if uncoloring modifications were applied to the structure and `false` otherwise.

OEUpdateMMCIFData

```
bool OEUpdateMMCIFData(OEMolBase& mol, const OECIFOptions& opts)
```

Updates MMCIF header metadata based on the structural data of the input molecule. If metadata already exists, it is replaced with the updated header.

See also

- *OEAddMMCIFData* function to add MMCIF data
- *OESetMMCIFData* function to set MMCIF data
- *mmCIF Metadata Manipulation* section

OEWeightedRMSD

Array-Based OEWeightedRMSD

```
double OEWeightedRMSD(const float* refcrds, const float* fitcrds,
                      const float* weightArray, unsigned int size,
                      bool overlay = false, double* rot = 0,
                      double* trans = 0)
double OEWeightedRMSD(const double* refcrds, const double* fitcrds,
                      const double* weightArray, unsigned int size,
                      bool overlay = false, double* rot = 0,
                      double* trans = 0)
```

Returns the weighted root mean squared deviation between two sets of Cartesian coordinates.

weightArray
: The array should be of length *size* and should contain the relative weight of each coordinate. The resulting RMSD is only affected by the relative weighting, so if all the weights are 1.0 or if all the weights are 2.0, the result will be the same.

Note

This function only differs from the function *OERMSD* by the additional parameter 'weightArray'. For details of the other parameters see *OERMSD* function.

Full Molecule-Based OEWeightedRMSD

```
double OEWeightedRMSD(const OEMolBase& ref, const OEMolBase& fit,
                      const double* weightArray,
                      bool automorph = true, bool heavyOnly = true,
                      bool overlay = false, double* rot = 0,
                      double* trans = 0)
bool OEWeightedRMSD(const OEMolBase& ref, const OEMCMolBase& fit,
                    const double* weightArray, double* rmsdArray,
                    bool automorph = true, bool heavyOnly = true,
                    bool overlay = false, double* rot = 0,
                    double* trans = 0)
```

Calculates the weighted root mean squared deviation between two molecules.

weightArray
: This array should be of length *fit.GetMaxAtomIdx()* and should contain the relative weight of each coordinate. The resulting RMSD is only affected by the relative weighting, so if all the weights are 1.0 or if all the weights are 2.0, the result will be the same.

Note

This function only differs from the function *OERMSD* by the additional parameter 'weightArray'. For details of the other parameters see *OERMSD* function.

Partial Molecule-Based OEWeightedRMSD

```
double OEWeightedRMSD(const OEMolBase& ref, const OEMolBase& fit,
                      const double* weightArray,
                      const OEMatchBase& match, bool overlay = false,
                      double* rot = 0, double* trans = 0)
bool OEWeightedRMSD(const OEMolBase& ref, const OEMCMolBase& fit,
                    const double* weightArray, double* rmsdArray,
                    const OEMatchBase& match, bool overlay = false,
                    double* rot = 0, double* trans = 0)
```

These functions are quite similar to the previous three. However, rather than considering automorphisms and heavy atoms, these functions allow a user to explicitly specify which substructure of the two molecules should be used to determine the RMSD. The match can be generated by hand, or with any of **OEChem TK**'s matching algorithms such as *OESubSearch* or *OEMCSSearch*.

weightArray
: This array should be of length *fit.GetMaxAtomIdx()* and should contain the relative weight of each coordinate. The resulting RMSD is only affected by the relative weighting, so if all the weights are 1.0 or if all the weights are 2.0, the result will be the same. The 'match' parameter takes precedence over the weights, so only weights that are part of the match will be relevant to the calculation.

Note

This function only differs from the function *OERMSD* by the additional parameter 'weightArray'. For details of the other parameters see *OERMSD* function.

See also

- *OERMSD* function
- *OEConfRMSD* class
- *Substructure Search* section
- *Maximum Common Substructure Search* section

OEWrite2DRingDictionary

```
bool OEWrite2DRingDictionary(oemolostream& oms,
                             const OE2DRingDictionary& ringdict)
bool OEWrite2DRingDictionary(OEPlatform::oeostream& os,
                             const OE2DRingDictionary& ringdict)
bool OEWrite2DRingDictionary(const std::string& filename,
                             const OE2DRingDictionary& ringdict)
```

Writes a ring template dictionary binary file.

See also

- *OE2DRingDictionary* class
- *2D Coordinate Generation* chapter

OEWriteCDXFile

```
bool OEWriteCDXFile(oemolostream& ofs, OEMolBase& mol)
```

Writes the given molecule to the output molstream in ChemDraw CDX file format.

OEWriteConstMolecule

```
unsigned OEWriteConstMolecule(oemolostream& ofs, const OEGraphMol& mol)
unsigned OEWriteConstMolecule(oemolostream& ofs, const OEMol& mol)
unsigned OEWriteConstMolecule(oemolostream& ofs, const OEQMol& mol)
unsigned OEWriteConstMolecule(oemolostream& ofs, const OEMolBase& mol)
unsigned OEWriteConstMolecule(oemolostream& ofs, const OEMCMolBase& mol
unsigned OEWriteConstMolecule(oemolostream& ofs, const OEQMolBase& mol)
```

These functions provide high-level file format writers for multiple file formats. These functions are built on top of **OEChem TK**'s low-level file format writers. If needed, these functions will make a copy of the molecule in order to automatically invoke the appropriate perception routines (aromaticity, atom typing, chirality perception, residue perception, etc...) as required by the selected file format. This greatly simplifies the task of writing molecules to a stream, but may provide less control than calling the lower-level functions directly.

Since these functions may make a copy of the molecule they'll generally be slower than the analogous *OEWriteMolecule* functions, which modify the molecule directly.

All of the overloads of *OEWriteConstMolecule* functions return an error code from the *OEWriteMolReturnCode* namespace. A return code of `0` or *OEWriteMolReturnCode.Success* indicates the write had no problems.

ofs

The output stream into which the molecule will be written.

The file format controlled by the format field of the *oemolostream* which may be set and retrieved using the methods *oemolstreambase.SetFormat*, and *oemolstreambase.GetFormat* respectively.

Additionally, the *OEWriteConstMolecule* functions provide some level of control over the processing applied to a molecule prior to writing using **OEChem TK**'s file format flavor mechanism. Each output molecule stream maintains an independent 'flavor' value for each output file format that can be specified by *oemolstreambase.SetFlavor*. These 'flavors' are unsigned integer values that are the combination of bit patterns specified by the *OEOFlavor* namespace. The interpretation of each bit in a flavor depends upon the output file format to which it refers, and enables or disables processing associated with writing that output file format.

mol

The molecule that is written into the stream.

See also

- *OEWriteMolReturnCode* namespace
- *the OEFormat namespace*
- *OEOFlavor* namespace
- *OEWriteMolecule* function
- Example program *CliqueAlign.cs*

OEWriteCSVFile

```
void OEWriteCSVFile(oemolostream& ofs, OEMolBase& mol,
                    unsigned int flavor = OEOFlavor::CSV::DEFAULT)
```

Write the molecule `mol` to the stream `ofs` in the *OEFormat.CSV* format as described in the *CSV File Format* section.

See also

OEOFlavor.CSV

OEWriteCIFFile

```
void OEWriteCIFFile(oemolostream& ofs, OEMolBase& mol,
                    const unsigned int flavor)
```

Writes the molecule as an CIF or mmCIF-file format file to the output stream 'ofs', depending on the content of the molecule. If *OEResidue* information is available on any atom in the molecule, it will be written by the mmCIF writer. The mmCIF writer will, additionally, switch between two modes, the mmCIF format for macromolecules or the Chemical Component Dictionary format. The Chemical Components Dictionary format is used when the molecule being written has only a single residue in it. A limited number of different CIF and mmCIF format variants are supported by the use of 'flavor' parameter from the *OEOFlavor.CIF* or *OEOFlavor.MMCIF* namespace.

OEWriteCSVHeader

```
void OEWriteCSVHeader(oemolostream& ifs, const OECSVHeader& header,
                      unsigned int flavor = OEOFlavor::CSV::DEFAULT)
```

Write out the column names specified by the *OECSVHeader* object into the stream passed as the `ifs` argument. Nothing will be written if *OEOFlavor.CSV.Header* is not set.

See also

CSV File Format

OEWriteFASTAFile

```
void OEWriteFASTAFile(oemolostream& ofs, OEMolBase& mol,
                      const char* ident)
```

Writes the molecule as a FASTA format protein sequence to the output stream 'ofs'. The 'ident' parameter is used to specify the sequence identifier to be written before the molecule's title on for each amino acid sequence.

OEWriteMacroModelFile

```
void OEWriteMacroModelFile(oemolostream& ofs, OEMolBase& mol)
void OEWriteMacroModelFile(oemolostream& ofs, OEMCMolBase& mol)
```

Writes the molecule as a MacroModel format file to the output stream 'ofs'.

OEWriteMDLFile

```
void OEWriteMDLFile(oemolostream& ofs, OEMolBase& mol,
                    unsigned int flavor = OEOFlavor::MDL::DEFAULT)
```

Writes the molecule as a MDL mol format file to the output stream 'ofs'. A number of different MDL format variants are supported by the use of the 'flavor' parameter from the *OEOFlavor.MDL* namespace.

By default, only molecules with more than 999 atoms or enhanced stereo groups will be written in V3000 format. The *OEOFlavor.MDL.MV30* flavor can be used to force all molecules to be written in the V3000 format.

OEWriteMol2File

```
void OEWriteMol2File(oemolostream& ofs, OEMolBase& mol,
                     bool m2h = true, bool substruct = false)
```

Writes the molecule as a Tripos mol2 format file to the output stream 'ofs'. The 'm2h' parameter specifies whether implicit hydrogens should be written out as explicit hydrogens in the mol2 file. If the value of 'm2h' is `false`, the mol2 file writer attempts to suppress writing hydrogens, if the resulting connection table can be read unambiguously.

OEWriteMolecule

```
unsigned int OEWriteMolecule(oemolostream& ofs, OEGraphMol& mol)
unsigned int OEWriteMolecule(oemolostream& ofs, OEMol& mol)
unsigned int OEWriteMolecule(oemolostream& ofs, OEQMol& mol)
unsigned int OEWriteMolecule(oemolothread& ofs, OEMolBase& mol)
```

(continues on next page)

(continued from previous page)

```
unsigned int OEWriteMolecule(oemolothread& ofs, OEMCMolBase& mol)
unsigned int OEWriteMolecule(oemolothread& ofs, OEQMolBase& mol)
```

These functions provide high-level file format writers for multiple file formats. These functions are built on top of **OEChem TK**'s low-level file format writers and may modify the molecule by automatically invoking the appropriate perception routines (aromaticity, atom typing, chirality perception, residue perception, etc…) as required by the selected output format. This greatly simplifies the task of writing molecules to a stream, but may provide less control than calling the lower-level functions directly.

There are a parallel set of *OEWriteConstMolecule* functions which write identical output without modifying the given molecule. These const functions will operate on a copy of the molecule if additional perception is required.

All of the overloads of *OEWriteMolecule* functions return an error code from the *OEWriteMolReturnCode* namespace. A return code of `0` or *OEWriteMolReturnCode.Success* indicates the write had no problems.

ofs

The output stream into which the molecule will be written.

The file format controlled by the format field of the *oemolostream* which may be set and retrieved using the methods *oemolostreambase.SetFormat*, and *oemolostreambase.GetFormat* respectively.

Additionally, the *OEWriteMolecule* functions provide some level of control over the processing applied to a molecule prior to writing using **OEChem TK**'s file format flavor mechanism. Each output molecule stream maintains an independent 'flavor' value for each output file format that can be specified by *oemolstreambase.SetFlavor*. These 'flavors' are unsigned integer values that are the combination of bit patterns specified by the *OEOFlavor* namespace. The interpretation of each bit in a flavor depends upon the output file format to which it refers, and enables or disables processing associated with writing that output file format.

mol

The molecule that is written into the stream.

See also

- *OEWriteMolReturnCode* namespace
- *the OEFormat namespace*
- *OEOFlavor* namespace
- *OEWriteConstMolecule* function
- Example program *Convert.cs*

OEWriteMolToBytes

```
OEIntArray OEWriteMolToByteArray(String format, OEMolBase mol)
OEIntArray OEWriteMolToByteArray(String format, OEMCMolBase mol)

byte[] OEWriteMolToBytes(String format, OEMolBase mol)
byte[] OEWriteMolToBytes(int format, int flavor, boolean gzip,
                         OEMolBase mol)

byte[] OEWriteMolToBytes(String format, OEMCMolBase mol)
byte[] OEWriteMolToBytes(int format, int flavor, boolean gzip,
                         OEMCMolBase mol)
```

Encodes a molecule in terms of 'format'. 'format' must be a file extension that is writeable by OpenEye, for example: ".oeb.gz". Overloads which take an unsigned 'format', unsigned 'flavor' and Boolean 'gzip' parameters expect the format or the format and flavor to be specified by one of the constants in *OEFormat* and *OEOFlavor*.

> **Warning**
>
> The overload that takes an OEIntArray is deprecated as of 2018.Feb and will be removed in a future version of **OEChem TK**, please use *OEWriteMolToBytes* instead.

```
OEMolBase mol = new OEGraphMol();
OEChem.OESmilesToMol(mol, "c1ccccc1");
byte[] bytes = OEChem.OEWriteMolToBytes(".oeb", mol);
```

```
OEMolBase mol = new OEGraphMol();
OEChem.OESmilesToMol(mol, "c1ccccc1");
byte[] bytes = OEChem.OEWriteMolToBytes(OEFormat.OEB, OEChem.
↪OEGetDefaultOFlavor(OEFormat.OEB), false, mol);
```

OEWriteMOPACInputFile

```
void OEWriteMOPACInputFile(oemolostream& ofs, OEMolBase& mol,
                           unsigned int flavor = OEOFlavor::MOPAC::DEFAULT)
```

Writes the molecule in MOPAC input file format to the output stream 'ofs' with 'flavor' parameter from the *OEOFlavor.MOPAC* namespace.

OEWritePDBFile

```
bool OEWritePDBFile(oemolostream& ofs, OEMolBase& mol,
                    unsigned int flavor = OEOFlavor::PDB::DEFAULT)
```

Writes the molecule as a Brookhaven PDB format file to the output stream 'ofs'. A number of different PDB format variants are supported by the use of 'flavor' parameter from the *OEOFlavor.PDB* namespace.

OEWriteXYZFile

```
void OEWriteXYZFile(oemolostream& ofs, const OEMolBase& mol,
                    bool charges = true, bool symbols = true,
                    bool doubleprecision = false)
```

Writes the molecule as an XMol XYZ format file to the output stream 'ofs'.

4.2 Preliminary OEChem API

4.2.1 Preliminary OEChem Classes

OEHasAminoAcidType

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEHasAminoAcidType
```

This functor can be used to identify monomers with the given amino acid type (*OEAminoAcidType*).

Constructors

```
OEHasAminoAcidType(unsigned aminoAcidType)
```

Initializes the functor with the given polymer type from the *OEAminoAcidType* namespace.

```
OEHasAminoAcidType(const OEHasAminoAcidType &)
```

Copy Constructors.

operator()

```
bool operator()(const OEMonomer &) const
```

Returns true, if the given monomer has the same amino acid type the *OEHasAminoAcidType* functor is initialized with.

CreateCopy

```
OESystem::OEUnaryFunction<OEMonomer, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasAminoAcidType* object is dynamically allocated and owned by the caller.

OEHasMonomerGroup

```
class  OEHasMonomerGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents *OEHasMonomerGroup* functor that identifies groups (*OEGroupBase*) that are representing a monomer specified by a code.

See also

- *OEIsMonomerGroup* for a functor that identifies groups that store monomer information.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasMonomerGroup::OEHasMonomerGroup(const std::string codeSet,
                                     const std::string code)
```

Constructor that takes a code set and a code to identify the monomer group.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasMonomerGroup* object is dynamically allocated and owned by the caller.

OEHasMonomerType

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEHasMonomerType
```

This functor can be used to identify monomers with the given monomer type (*OEMonomerType*).

Constructors

```
OEHasMonomerType(unsigned polymerType)
```

Initializes the functor with the given polymer type from the *OEMonomerType* namespace.

```
OEHasMonomerType(const OEHasMonomerType &)
```

Copy constructor.

operator()

```
bool operator()(const OEMonomer &) const
```

Returns true, if the given monomer has the same monomer type the *OEHasMonomerType* functor is initialized with.

CreateCopy

```
OESystem::OEUnaryFunction<OEMonomer, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasMonomerType* object is dynamically allocated and owned by the caller.

OEHasPolymerType

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEHasPolymerType
```

This functor can be used to identify monomers with the given polymer type (*OEPolymerType*).

Constructors

```
OEHasPolymerType(unsigned polymerType)
```

Initializes the functor with the given polymer type from the *OEPolymerType* namespace.

```
OEHasPolymerType(const OEHasPolymerType &)
```

Copy constructor.

operator()

```
bool operator()(const OEMonomer &) const
```

Returns true, if the given monomer has the same polymer type the *OEHasPolymerType* functor is initialized with.

CreateCopy

```
OESystem::OEUnaryFunction<OEMonomer, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasPolymerType* object is dynamically allocated and owned by the caller.

OEHelmGenerationOptions

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEHelmGenerationOptions
```

Encapsulates the properties that control HELM generation from a molecule.

> **See also**
>
> - *OEHelmGenerationResult* class

Constructors

```
OEHelmGenerationOptions()
OEHelmGenerationOptions(const std::string& codeSet,
                        bool allowUnspecifiedStereo = false,
                        bool allowPartialMonomerMatch = false)
```

Default constructor.

```
OEHelmGenerationOptions(const OEHelmGenerationOptions &)
```

Copy constructor.

GetAllowUnmatchedFragments

```
bool GetAllowUnmatchedFragments() const
```

Returns whether unmatched fragments are allowed when generating the HELM when using *OEMolToHelm* function. Unmatched fragments will be encoded as embedded SMILES in the generated HELM string.

GetAllowUnspecifiedStereo

```
bool GetAllowUnspecifiedStereo() const
```

Returns whether unspecified atom or bond stereochemistry are allowed when generating the HELM when using *OEMolToHelm* function.

GetCodeSet

```
std::string GetCodeSet() const
```

Returns the code-set to be used for generating the HELM string.

See also

- *OEHelmGenerationOptions.SetCodeSet* method

GetMinUnmatchedAtomsRatio

```
double GetMinUnmatchedAtomsRatio() const
```

Returns the minimum atom to fragment ratio that are allowed when generating the HELM when using *OEMolToHelm* function.

SetAllowUnmatchedFragments

```
bool SetAllowUnmatchedFragments(bool allow)
```

Sets whether unmatched fragments are allowed when generating the HELM when using *OEMolToHelm* function.

SetAllowUnspecifiedStereo

```
bool SetAllowUnspecifiedStereo(bool allow)
```

Sets whether unspecified atom or bond stereochemistry are allowed when generating the HELM when using *OEMolToHelm* function.

SetCodeSet

```
void SetCodeSet(const std::string& codeSet)
```

Sets the code-set to be used for generating the HELM string. If not set the primary code-set will be used.

See also

- *OEMonomerSet.GetPrimaryCodeSet* method

SetMinUnmatchedAtomsRatio

```
bool SetMinUnmatchedAtomsRatio(double minRatio)
```

Sets the minimum atom to fragment ratio that are allowed when generating the HELM when using *OEMolToHelm* function.

OEHelmGenerationResult

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEHelmGenerationResult
```

This class is used to return additional information when using the *OEMolToHelm* function.

Constructors

```
OEHelmGenerationResult()
OEHelmGenerationResult(const OEHelmGenerationResult &)
```

Default and copy constructor.

GetCodeSet

```
std::string GetCodeSet() const
```

Returns the name of the code-set that is used to successfully generate the HELM string.

GetReturnCode

```
unsigned GetReturnCode() const
```

Returns a code from the *OEHelmGenerationReturnCode* namespace.

GetUnmappedFragments

Returns the SMILES representation of unmapped fragments in the molecule.

```
std::vector<std::string> GetUnmappedFragments() const
```

GetWarning

```
std::string GetWarning() const
```

If the HELM generation fails, i.e. the return code is not *OEHelmGenerationReturnCode.Success*, this message provides details about the reason for the failure.

OEHelmParsingResult

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEHelmParsingResult
```

This class is used to return additional information when using the *OEHelmToMol* function.

Constructors

```
OEHelmParsingResult()
OEHelmParsingResult(const OEHelmParsingResult &)
```

Default and copy constructor.

GetCodeSet

```
std::string GetCodeSet() const
```

Returns the name of the code-set that is used to successfully parse the HELM string.

GetErrorPosition

```
unsigned GetErrorPosition() const
```

If the HELm parsing fails, i.e. the return code is not *OEHelmParsingReturnCode.Success*, this shows the position in the HELM string where the failure occurs.

GetReturnCode

```
unsigned GetReturnCode() const
```

Returns a code from the *OEHelmParsingReturnCode* namespace.

GetWarning

```
std::string GetWarning() const
```

If the HELM parsing fails, i.e. the return code is not *OEHelmParsingReturnCode.Success*, this message provides details about the reason for the failure.

OEIsInMonomerCodeSet

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEIsInMonomerCodeSet
```

Functor that identifies monomers whose code is defined in the specified code set.

Constructors

```
OEIsInMonomerCodeSet(const std::string& codeSet)
```

Constructs the functor with the name of the code set to match against.

```
OEIsInMonomerCodeSet(const OEIsInMonomerCodeSet &)
```

Copy constructor.

operator()

```
bool operator()(const OEMonomer &) const
```

Returns `true` if the monomer has a code defined in the code set specified at construction.

CreateCopy

```
OESystem::OEUnaryFunction<OEMonomer, bool>* CreateCopy() const
```

OEIsMonomerGroup

```
class  OEIsMonomerGroup : public OESystem::OEUnaryPredicate<OEGroupBase>
```

This class represents *OEIsMonomerGroup* functor that identifies groups (*OEGroupBase*) that store monomer information.

See also

- *OEGroupType* for built-in group types.
- *OEHasMonomerGroup* for a functor that identifies groups that match a monomer group by code set and code.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsMonomerGroup()
```

Default constructor.

CreateCopy

```
OESystem::OEUnaryFunction<OEGroupBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsMonomerGroup* object is dynamically allocated and owned by the caller.

OEMonomer

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEMonomer : public OESystem::OEBase
```

Stores validated and pre-computed information for a monomer to be used as a building block for representing biological macromolecules.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEMonomer(const OEMonomerData &)
```

Initializes a monomer from *OEMonomerData* object. If the *OEMonomerData* is not valid an error will be thrown and the the generated *OEMonomer* object will be invalid too.

```
OEMonomer(const OEMonomer &)
```

Copy constructor.

GetAminoAcidType

```
unsigned GetAminoAcidType() const
```

Returns the amino acid of the monomer.

See also

- *OEAminoAcidType* namespace

GetCanonicalSmiles

```
std::string GetCanonicalSmiles() const
```

Returns the canonical smiles of the monomer without the R-group definitions. In case of alanine, that is defined with `[H:1]N[C@@H](C)C(=O)[OH:2]`, its canonical smiles will be `C[C@@H](C(=O)O)N`.

See also

- *OEMonomer.GetSmiles* method

GetCode

```
std::string GetCode(const std::string& codeSet) const
```

Returns the code of the monomer in the given code set. Empty string is returned if the monomer has no code specified.

GetCodes

```
const std::map<std::string, std::string>& GetCodes() const
```

Returns all the code/code-set pairs defined for the monomer.

GetMonomerType

```
unsigned GetMonomerType() const
```

Returns the monomer type of the monomer.

See also

- *OEMonomerType* namespace

GetName

```
std::string GetName() const
```

Returns the full name of the monomer, or an empty string if the name was not specified at initialization.

GetPolymerType

```
unsigned GetPolymerType() const
```

Returns the polymer type of the monomer.

See also

- *OEPolymerType* namespace

GetSmiles

```
std::string GetSmiles() const
```

Returns the smiles representation of the monomer that contains the atom map indices representing connection points. Such as `[H:1]N[C@@H](C)C(=O)[OH:2]` defining alanine.

See also

- *OEMonomer.GetCanonicalSmiles* method

HasCode

```
bool HasCode(const std::string& codeSet) const
```

Return whether the monomer has a code defined in the given code-set.

HasRGroup

```
bool HasRGroup(unsigned idx) const
```

Returns whether the monomer has a specific (R-groups) connections.

IsCTerminal

```
bool IsCTerminal() const
```

Returns whether a monomer can be used at the C-terminal of a peptide chain.

IsNTerminal

```
bool IsNTerminal() const
```

Returns whether a monomer can be used at the N-terminal of a peptide chain.

IsValid

```
bool IsValid() const
```

Returns whether the monomer was initialized successfully.

NumCodes

```
unsigned NumCodes() const
```

Returns the number of codes the monomer is defined in various code-sets.

NumRGroups

```
unsigned NumRGroups() const
```

Returns the number of connections (R-groups) of the monomer.

OEMonomerData

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEMonomerData
```

Stores the basic definition required for a monomer (*OEMonomer*) to be used as a building block for representing biological macromolecules.

Table 42: **Example of monomer definition of L-Alanine**

Property	Value	Required/Optional/Generated
Smiles	`[H:1]N[C@@H](C)C(=O)[OH:2]`	required
Canonical Smiles	`C[C@@H](C(=O)O)N`	generated
Name	`L-Alanine` (optional)	optional
Polymer type	*OEPolymerType.Peptide*	required
Monomer type	*OEMonomerType.Backbone*	required
Monomer code-set/code pair	`A` in code-set `Standard`	required for at least one code-set

See also

- *OEIsValidMonomerData* function
- *OEMonomerValidationResult* class

Constructors

```
OEMonomerData()
```

Default constructor that creates an *OEMonomerData* object.

```
OEMonomerData(const std::string& smiles,
              unsigned polymerType = OEPolymerType::Peptide,
              unsigned monomerType = OEMonomerType::Backbone)
```

Constructor that initializes an *OEMonomerData* object.

See also

- *OEPolymerType* namespace
- *OEMonomerType* namespace

```
OEMonomerData(const OEMonomerData &)
```

Copy constructor.

AddCode

```
void AddCode(const std::string& codeSet, const std::string& code)
```

Adds a code to a monomer for a given code-set.

GetCode

```
std::string GetCode(const std::string& codeSet) const
```

Returns the code of the monomer for the given code-set. Returns an empty string if there is no code defined for the code-set.

GetCodes

```
const std::map<std::string, std::string>& GetCodes() const
```

Returns the map that contains all the code-set and code pairs defined for the monomer.

GetMonomerType

```
unsigned int GetMonomerType() const
```

Returns the monomer type of the monomer.

See also

- *OEMonomerType* namespace

GetName

```
std::string GetName() const
```

Returns the full name of the monomer. It is optional to define and only used to reference name.

GetPolymerType

```
unsigned int GetPolymerType() const
```

Returns the polymer type of the monomer.

See also

- *OEPolymerType* namespace

GetSmiles

```
std::string GetSmiles() const
```

Returns the smiles representation of the monomer that contains the atom map indices representing connection points. Such as `[H:1]N[C@@H](C)C(=O)[OH:2]` defining alanine.

GetCanonicalSmiles

```
std::string GetCanonicalSmiles() const
```

Returns the canonical smiles representation of the monomer (without atom mapping) Such as `C[C@@H](C(=O)O)N]` for alanine.

HasCode

```
bool HasCode(const std::string& codeSet) const
```

Returns whether the monomer has a code defined for the given code set.

IsValid

```
bool IsValid() const
```

Returns whether the *OEMonomerData* object is fully defined and correct.

See also

- *OEIsValidMonomerData* function

NumCodes

```
unsigned int NumCodes() const
```

Returns the number of codes defined for the monomer in various code-sets.

SetCodes

```
void SetCodes(const std::map<std::string, std::string>& codes)
```

Sets a code of the monomer for a specific code-set.

SetMonomerType

```
void SetMonomerType(unsigned int)
```

Sets the monomer type of the monomer.

See also

- *OEMonomerType* namespace

SetName

```
void SetName(const std::string &)
```

Sets the name of the monomer. The name can be arbitrary, but it is recommended use the IUPAC name or any other name that well the monomer. For example, the name of the `C[C@@H](C(=O)O)N]` monomer of can be `Alanine` , `L-alanine` or `L-alpha-Alanine` or `(2S)-2-aminopropanoic acid`. Setting the name is optional.

SetPolymerType

```
void SetPolymerType(unsigned int)
```

Sets the polymer type of the monomer.

See also

- *OEPolymerType* namespace

SetSmiles

```
void SetSmiles(const std::string &)
```

Sets the smiles of the monomer. The given smiles has to define possible attachment points of the monomer using map indices. For example, `[H:1]N[C@@H](C)C(=O)[OH:2]` defining alanine with two connection points to from a peptide bonds with adjacent amino acid monomers.

See also

- *OEPerceivePeptideMonomerConnections* function

Validate

```
bool Validate() const
```

Calls the *OEIsValidMonomerData* function to validate the monomer data object.

OEMonomerSet

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEMonomerSet
```

A container class that stores monomer definitions.

There are two built in monomer set supported by OEChem

- "Standard" - contains 20 standard amino acid in the code-set named *Standard*
- "OpenEye" - contains more than 200 diverse monomers in defined in four code-sets named: `OpenEye`, `Standard`, `PDB` and `ChEMBL`

A monomer has to be defined in at least one code-set. In a code-set each monomer has to have a unique name, built different code set can a monomer cna be named differently.

Table 43: **Example of monomer codes in the built-in OpenEye monomer-set**

Monomer Name	Code in *OpenEye*[1]	Code in *Standard*	Code in *PDB*	Code in *ChEMBL*
Alanine	*A*	*A*	*ALA*	*A*
Cysteine	*C*	*C*	*CYS*	*c*
D-Alanine	*dAla*		*DAL*	*dA*
(2S)-2-amino-3-(3-pyridyl)propanoic acid	*3Pal*		*1MH*	*3-Pal*

The following example shows how to generate a custom monomer set:

The generated *OEMonomerSet* object can be saved into a JSON file using *OEWriteMonomerSet* and later loaded with thee *OEReadMonomerSet* function.

See also

- *OEPerceivePeptideMonomerConnections* function
- *OEGetPolymerType* and *OEGetMonomerType* functions
- *OEMonomerData* class and *OEIsValidMonomerData* function

OpenEye Python Cookbook Code Example

- Generating Custom Monomer Set **OpenEye Python Cookbook** recipe

Constructors

```
OEMonomerSet()
```

Constructors an empty monomer set.

AddCodeSet

```
bool AddCodeSet(const std::string& codeSet)
```

Adds a code-set name.

AddMonomer

```
bool AddMonomer(const OEMonomerData &)
```

Adds a monomer (defined by in a *OEMonomerData* object) to the monomer set. The monomer is validated with the *OEIsValidMonomerData* function. The method will returns if the monomer was valid and it was successfully added to the set, otherwise it will return false.

Adding a valid monomer (*OEMonomerData*) can fail if:

[1] OpenEye code-set following naming-convention of [Ghiandoni-2026]

- the monomer is already present in the set (using canonical isomeric smiles for equivalency check)
- the monomer has a code that is already used in the specific code-set stored in the set

GetCodeSets

```
std::vector<std::string> GetCodeSets() const
```

Returns all the code-sets supported by the monomer set.

See also

- *OEMonomerSet.GetPrimaryCodeSet* method

GetCodes

```
std::vector<std::string> GetCodes(const std::string& codeSet) const
```

Returns all the monomer codes stored in the monomer set for the given code-set.

GetMonomer

```
const OEMonomer* GetMonomer(const std::string& codeSet,
                            const std::string& code) const
```

Returns a monomer pointer with the given code in the specified code-set. NULL pointer will be return if no such monomer exists in the set.

GetMonomers

```
OESystem::OEIterBase<const OEMonomer>* GetMonomers() const
```

Returns an iterator over all *OEMonomer* objects stored in monomer set.

```
OESystem::OEIterBase<const OEMonomer>* GetMonomers(const OESystem::OEUnaryPredicate
↪<OEMonomer>& pred) const
```

Returns an iterator over all *OEMonomer* objects stored in monomer set that satisfies the given predicate.

GetPrimaryCodeSet

```
std::string GetPrimaryCodeSet() const
```

Returns the name of the primary code-set of the monomer set. The primary code-set is the set that has the largest number of monomers defined with a unique code.

See also

- *OEMonomerSet.GetCodeSets* method
- *OEMonomerSet.NumMonomers* method

GetVersion

```
std::string GetVersion() const
```

Returns the version of the monomer set.

See also

- *OEMonomerSet.SetVersion* method

HasCodeInCodeSet

```
bool HasCodeInCodeSet(const std::string& codeSet,
                      const std::string& code) const
```

Returns whether the monomer set contains a specific monomer with the given code and code-set.

HasCodeSet

```
bool HasCodeSet(const std::string& codeSet) const
```

Returns whether the monomer set contains any monomer for the given code-set.

HasMonomer

```
bool HasMonomer(const OEMonomerData &) const
```

Returns whether the monomer (*OEMonomerData*) already exists in the monomer set. Each monomer in the *OEMonomerSet* has to have a unique canonical smiles.

IsValid

```
bool IsValid() const
```

Returns whether the monomer set stores at least one valid monomer object.

NumCodeSets

```
unsigned NumCodeSets() const
```

Returns the number of unique code set the monomers are defined.

NumMonomers

```
unsigned NumMonomers() const
```

Returns the number of monomers in the container.

```
unsigned NumMonomers(const std::string& codeSet) const
```

Returns the number of monomers in the container that associated wit a code in the given code-set.

See also

- *OEMonomerSet.GetPrimaryCodeSet* method

```
unsigned NumMonomers(const OESystem::OEUnaryPredicate<OEMonomer>& pred) const
```

Returns the number of monomers that satisfy the given predicate.

See also

- *OEHasAminoAcidType* predicate
- *OEHasMonomerType* predicate
- *OEHasPolymerType* predicate
- *OEIsInMonomerCodeSet* predicate

SetVersion

```
void SetVersion(const std::string& version)
```

Sets the version of the monomer set. The function does not validate the format of the given string. However OpenEye's built-in monomer sets are using the following conventions:

The version format is `A.B.C` (The first is version is "1.0.0")

- A: incremented for **major** changes in monomer set that has effect of parsing already generated HELMS with the given code set.
  - monomer is removed
  - monomer smiles has been corrected (for example correcting stereo configuration)
  - monomer code is modified
- B: incremented for **minor** changes in monomer set that has effect on generating new HELMs but will not effect paring of HELMs generated with previous versions (backwards-compatibility)
  - new monomer is added
  - new R-group is added to already existing monomer
- C: incremented for **micro** changes hat does not effect either HELM parsing or generation (backwards-compatibility)
  - monomer name change
  - adding additional monomer properties (such as associating colors with monomers for depiction)

OEMonomerValidationResult

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OEMonomerValidationResult
```

This class is used to return additional information when using the *OEIsValidMonomerData* function.

Constructors

```
OEMonomerValidationResult()
OEMonomerValidationResult(const OEMonomerValidationResult &)
```

Default and copy constructor.

GetReturnCode

```
unsigned GetReturnCode() const
```

Returns a code from the *OEMonomerValidationReturnCode* namespace.

GetWarning

```
std::string GetWarning() const
```

If the monomer validation fails, i.e. the return code is not *OEMonomerValidationReturnCode.Success*, this message provides details about the reason for the failure.

OERxnAutomapper

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OERxnAutomapper
```

Applies reactant and product reaction mapping correspondence information to a reaction molecule.

See also

- *OERxnAutomapperOption* class

Constructors

```
OERxnAutomapper()
```

Default constructor that initializes an *OERxnAutomapper* object.

```
OERxnAutomapper(const OERxnAutomapperOptions& opts)
```

Constructor that initializes an *OERxnAutomapper* object with the provided option class settings.

```
OERxnAutomapper(const OERxnAutomapper& rhs)
```

Copy constructor.

AutoMapRxn

```
unsigned AutoMapRxn(OEMolBase& rxn)
```

Requests the provided reaction be processed to apply atom-atom mapping correspondence.

See also

- *OERxnAutomapStatus* namespace

GetOptions

```
const OERxnAutomapperOptions& GetOptions() const
OERxnAutomapperOptions& GetOptions()
```

Returns the internal option class that controls the automap activities.

OERxnAutomapperOptions

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
class OERxnAutomapperOptions
```

Encapsulates the properties that control reaction mapping using *OERxnAutomapper*.

See also

- *OERxnAutomapper* class

Constructors

```
OERxnAutomapperOptions()
```

Default constructor that initializes an *OERxnAutomapperOptions* object with the following properties:

Table 44: **Default parameters of OERxnAutomapperOptions**

Description	Option	Default value
Mapping Method	*GetMappingMethod()*	*OERxnAutomapMethod.MCSMapping*
MCS Mapping Mode	*GetMCSExhaustiveMode()*	false
Clear Mapping	*GetClearMapping()*	true
Extend Mapping	*GetExtendMappedRegions()*	true
Strict Mapping	*GetStrictMapping()*	false

```
OERxnAutomapperOptions(const OERxnAutomapperOptions& rhs)
```

Copy constructor.

GetClearMapping

```
bool GetClearMapping() const
```

Returns whether existing atom-atom mappings should be cleared prior to attempting the reaction mapping activity. Existing atom-atom mappings *may* be used to guide the reaction mapping activity if not cleared.

See also

- *OEAtomBase.GetMapIdx* method
- *OEAtomBase.SetMapIdx* method

GetExtendMappedRegions

```
bool GetExtendMappedRegions() const
```

Returns whether unassigned mapped regions can be augmented with unambiguous reactant and product atom correspondences to maximize the size of the reaction mapping.

GetMappingMethod

```
unsigned GetMappingMethod() const
```

Returns the requested mapping method to use for the reaction mapping correspondence.

GetMCSExhaustiveMode

```
bool GetMCSExhaustiveMode() const
```

Returns whether the exhaustive MCS mapping mode should being used when the automapping is using the MCS method.

GetStrictMapping

```
bool GetStrictMapping() const
```

Returns the requested activation state for strict reaction mapping validation activities.

SetClearMapping

```
bool SetClearMapping(bool flag)
```

Sets whether existing atom-atom mappings should be cleared prior to attempting the reaction mapping activity. Existing atom-atom mappings may be used to guide the reaction mapping activity if not cleared.

See also

- *OERxnAutomapperOptions.GetClearMapping* method

SetExtendMappedRegions

```
bool SetExtendMappedRegions(bool flag)
```

Sets whether unassigned mapped regions can be augmented with unambiguous reactant and product atom correspondences to maximize the size of the reaction mapping.

SetMappingMethod

```
bool SetMappingMethod(unsigned method)
```

Sets the requested mapping method to use for the reaction mapping correspondence. This is simply a placeholder for future methods, currently only an MCS mapping approach is available.

SetMCSExhaustiveMode

```
bool SetMCSExhaustiveMode(bool flag)
```

Sets whether the slower exhaustive MCS mapping mode should be used when the automapping is using the MCS method.

SetStrictMapping

```
bool SetStrictMapping(bool flag)
```

Enables or disables strict reaction mapping validation activities.

4.2.2 Preliminary OEChem Functions

OEAminoAcidTypeToString

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::string OEAminoAcidTypeToString(unsigned)
```

Converts the values of the *OEAminoAcidType* namespace into string representation.

See also

- *OEAminoAcidType* namespace

OEApplyStateFromRef

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

This function can be used to change the tautomer state or a fix a broken valence state in a 3D molecule, based on a reference state (not required to be 3D). The function matches the substructures and transfers the state of the reference, like bond-orders, formal charge assignments, and hydrogen assignments to the input molecule. The input molecule can be smaller than the reference molecule, as this is intended to work for small molecule crystal structures where part of the molecule can have been degraded in the experiment. It also works for molecules that have been covalently bound to a protein, such that there is an R-group in place of the broken covalent bond. Due to symmetries in the substructure graph match, that are not unique in 3D, the function returns an iterator of output molecules where the state of the reference has been applied. An simple example would be a 3D molecule with two carboxylic acid groups on either side of a beneze ring. If the state being applied has one neutral carboxylic acid, and one negatively charged, two output molecules would have to be generated. While they are symmetric, if the 3D molecule is bound in a protein pocket, the two groups are not identical and one very likely fits the local environment better than the other, which will need to be checked.

Note

The state of hydrogens being implicit or explicit in the output will match that of the state of the reference molecle.

Note

The function currently ignores chirality of the two molecules, since this cannot be changed in a 3D molecule without conformational changes. However, the function will attempt to transfer chiral information on atoms and bonds from the reference state by setting the *OEAtomBase.SetChiral* flag to match the *OEAtomBase.HasStereoSpecified* flag from the reference state. The user is expected to call *OE3DToAtomStereo* to convert the chiral flag to atoms with specified stereo on 3D molecules.

```
OESystem::OEIterBase<OEMolBase>* OEApplyStateFromRef(const OEMolBase& input,
                                                     const OEMolBase& refMol)
OESystem::OEIterBase<OEMolBase>* OEApplyStateFromRef(const OEMolBase& input,
                                                     const std::string&␣
↪refSmiles)
```

The SMILES version uses *OEMolToSmiles* to convert the incoming smiles to a molecule usable for the substructure search.

See also

- *OEMolToSmiles* function
- *OESubSearch* function

OECreateAtomSmartsString

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OECreateAtomSmartsString(std::string& str,
                              const OEChem::OEAtomBase* atm,
                              unsigned int atomSMARTSFlags =␣
↪OESMARTSAtomFlag::AtomDefault,
                              bool bExplicitDelim = true)
```

For a specified atom, generate a SMARTS string of query qualifiers based on the specified atom's environment and using the constraints defined by the *OESMARTSAtomFlag* namespace. The `bExplicitDelim` argument indicates if the generated property list should be explicitly `;` delimited in the output to simplify post-processing for custom constraint generations.

This function is only provided in C++ and python.

OECreateBondSmartsString

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OECreateBondSmartsString(std::string& str,
                              const OEChem::OEBondBase* bnd,
                              unsigned int bondSMARTSFlags =␣
↪OESMARTSBondFlag::BondDefault,
                              bool bExplicitDelim = false)
```

For a specified bond, generate a SMARTS string of query qualifiers based on the specified bond's environment and using the constraints defined by the *OESMARTSBondFlag* namespace.

This function is only provided in C++ and python.

OECreatePeptideMonomerConnectionSmiles

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::string OECreatePeptideMonomerConnectionSmiles(const std::string& monomerSmiles)
```

The function identifies the backbone of the amino acid represented in the input string and marks the backbone connections in the output smiles:

- the hydrogen of the amino group of the backbone will be have atom mapping `1`
- the oxygen of the amino acid of the backbone with be have atom mapping `2`

For example, in case of the input smiles `C[C@@H](C(=O)O)N`, the function will return `[H:1]N[C@@H](C)C(=O)[OH:2]`.

If the input smiles does not represent an amino-acid or its backbone can not be identified, and empty string will be returned.

The current implementation supports the identification of backbone connection for alpha-, beta-, gamma-, and delta-amino acids. However, it does not identify connection point(s) on the side chain of the amino acid.

For example, in case of the input smiles `C([C@@H](C(=O)[OH:2])N)S` representing *cystine*, the function will return `[H:1]N[C@@H](CS)C(=O)[OH:2]"`, rather than `[H:1]N[C@@H](CS[H:3])C(=O)[OH:2]` where *R3* could be used to form disulfide cross-links.

When identifying the backbone of an amino acid, backbone is always selected as the shortest path between identified amino and carboxylic acid groups.

For example, in case of the input smiles `C(CC(=O)O)[C@@H](C(=O)O)N` representing *glutamic acid*, the function will return `[H:1]N[C@@H](CCC(=O)O)C(=O)[OH:2]`, rather than `[H:1]N[C@@H](CCC(=O)[OH:3])C(=O)[OH:2]` where *R3* could be used to form isopeptide bond.

When creating a custom monomer set, side-chain interactions of amino acids, along with the identification of connection points for C- and N-terminal monomers and generic non-peptidic monomers, must be specified manually.

> **See also**
>
> - *OEPerceivePeptideMonomerConnections* function
> - *OEMonomerConnectionSmilesFromCXSmiles* function

OECreateSmartsString

> **Attention**
>
> This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
void OECreateSmartsString(std::string& str, const OEMolBase& mol,
                          const char* smartsTag = nullptr,
                          unsigned int flavor = (OESMILESFlag::DEFAULT & ~
↪OESMILESFlag::Canonical),
```

(continues on next page)

(continued from previous page)

```
                          std::pair<const OEAtomBase*, const OEAtomBase*>* atmord =␣
↪nullptr)
```

Generate the SMARTS string from the provided molecule using the flavors from the *OESMILESFlag* namespace. If the `smartsTag` argument is null, the data tag applied by *OEParseSmarts* is assumed to contain the atom and bond annotations from the SMARTS parse, otherwise an explicit generic data tag can be provided for custom SMARTS constraint annotations. Note that custom constraint annotations are not validated, so extra checking for a valid SMARTS query will be required (e.g., by calling *OEParseSmarts*).

The provided default flavor for the SMARTS generation suppresses reordering which can invalidate the statically captured pattern constraints. Any explicitly provided non-default flavor should also suppress reordering activity for the same reason. While the flavor is provided for user control and experimentation, be aware that only actual atom/bond properties on the molecule are considered for the flavor provided, so any SMARTS qualifiers from the `smartsTag` generic data are considered to be completely irrelevant during canonicalization. Extra care is also required if any atom (*OESMARTSAtomFlag.Chirality*) or bond (*OESMARTSBondFlag.Chirality*) stereo parity is encoded as into the SMARTS string, requiring one or both of *OESMILESFlag.AtomStereo* for tetrahedral stereo parity SMARTS encodings, or *OESMILESFlag.BondStereo* for cis/trans SMARTS encodings.

See also

- *SMARTS*
- *OECreateAtomSmartsString*
- *OECreateBondSmartsString*

This function is only provided in C++ and python.

OEDetectMonomers

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEDetectMonomers(OEMolBase& mol, const OEMonomerSet& monomers)
bool OEDetectMonomers(OEMolBase& mol, const OEMonomerSet& monomers,
                      const std::string& codeSet)
```

Detects the monomers defined in the specified monomer set in the given molecule. The function returns true if all atoms in the molecule are part of the defined monomers and false otherwise.

mol
: The molecule (*OEMolBase*) to be searched for monomers.

monomers
: The monomer set (*OEMonomerSet*) that stores the monomers that are being detected.

code-set
: The code set to use for detecting monomers, if not specified the primary code-set of the monomer set will be used.

The detected monomers are stored on the molecule as groups (*OEGroupBase*).

Example

See also

- *OEMonomerSet* class
- *OEMolToHelm* function

OEGetAminoAcidType

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
unsigned int OEGetAminoAcidType(const OEMolBase& monomer_mol)
unsigned int OEGetAminoAcidType(const std::string& monomerSmiles)
```

Determines the amino acid type of the monomer defined by its molecule or smiles representation.

See also

- *OEAminoAcidType* namespace

OEGetHelmChainNames

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::vector<std::string> OEGetHelmChainNames(const std::string& helm)
```

Returns the chain names present in the given HELM string.

For example, in case of input `"PEPTIDE1{F.A}|CHEM1{[Fmoc]}$PEPTIDE1,CHEM1,1:R1-1:R1$$$"` it will return `PEPTIDE1` and `CHEM1`.

Warning

The *OEGetHelmChainNames* function does not perform full validation or parsing of the HELM string. It is intended for quickly extracting information from an already validated HELM.

See also

- *OEGetHelmMonomerCodes* function
- *OEHelmToMol* function

OEGetHelmMonomerCodes

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::vector<std::string> OEGetHelmMonomerCodes(const std::string& helm)
std::vector<std::string> OEGetHelmMonomerCodes(const std::string& helm,
                                               const std::string& chainName)
```

Returns the monomer codes present in the given HELM string.

For example, in case of input `"PEPTIDE1{F.A}|CHEM1{[Fmoc]}$PEPTIDE1,CHEM1,1:R1-1:R1$$$"` it will return `F`, `A` and `Fmoc`. If a chain name is specified only monomer codes present in that chain are returned. For example, in case of `PEPTIDE1` it will only return `F` abd `A`.

Warning

The *OEGetHelmMonomerCodes* function does not perform full validation or parsing of the HELM string. It is intended for quickly extracting information from an already validated HELM.

See also

- *OEGetHelmChainNames* function
- *OEHelmToMol* function

OEGetMonomerType

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
unsigned int OEGetMonomerType(const OEMolBase& monomer_mol)
unsigned int OEGetMonomerType(const std::string& monomerSmiles)
```

Determines the monomer type of the monomer defined by its molecule or smiles representation. Returns *OEMonomerType.Undefined* if given monomer does not satisfy the requirement of any of the specific monomer type.

See also

- *OEMonomerType* namespace

OEGetPolymerType

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
unsigned int OEGetPolymerType(const OEMolBase& monomer_mol)
unsigned int OEGetPolymerType(const std::string& monomerSmiles)
```

Determines the polymer type of the monomer defined by its molecule or smiles representation. Returns *OEPolymerType.Undefined* if given monomer does not satisfy the requirement of any of the specific polymer type.

See also

- *OEPolymerType* namespace

OEGetStandardAnalog

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
unsigned int OEGetStandardAnalog(const std::string& monomerSmiles)
```

Returns the residue index of the standard amino acid (from the *OEResidueIndex* namespace) that are most similar to monomer represented by the given SMILES, or *OEResidueIndex.UNK* if the monomer is not considered an analog.

See also

- *OEResidueIndex* namespace
- *OEGetAminoAcidCode* function

OEIsStandardMonomer

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEIsStandardMonomer(const std::string& monomerCanSmiles)
```

Returns true for the canonical smiles of the 20 standard amino acids.

See also

- *OEGetStandardAnalog* function

OEHelmToMol

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEHelmToMol(OEMolBase& mol, const std::string& helm,
                 const OEMonomerSet& monomers)
bool OEHelmToMol(OEMolBase& mol, const std::string& helm,
                 const OEMonomerSet& monomers,
                 OEHelmParsingResult& result)
```

Converts a HELM string into a molecule. The function will return true if the HELM string can be parsed and interpreted successfully, otherwise the function will return false and a given molecule will be cleared.

mol
The constructed molecule (*OEMolBase*).

helm
The HELM string that is parsed into the molecule.

monomers
The monomer set (*OEMonomerSet*) that stores the monomers that are used to interpret the HELM string.

If the given monomer set supports multiple code sets, *OEHelmToMol* automatically selects the code set that contains all monomer codes needed to parse the HELM string.

result
Is specified, the *OEHelmParsingResult* object will store additional parsing information.

After constructing the molecule the following properties are perceived:

- Rings are perceived by calling the *OEFindRingAtomsAndBonds* function.
- The aromaticity is assigned by calling the *OEAssignAromaticFlags* function using the *OEAroModel.OpenEye* aromaticity model.
- The chirality is perceived by calling the *OEPerceiveChiral* function.

Warning

The current implementation of HELM parsing algorithm does not support RNA polymers.

Example

OEHelmToMol can parse HELM strings with embedded SMILES in various styles. The examples below illustrate valid representations of eledoisin:

```
PEPTIDE1{[N1[C@@H](CCC1=O)C([R2])=O].P.S.K.D.A.F.I.G.L.M.[am]}$$$$
PEPTIDE1{[N1[C@@H](CCC1=O)C([*:2])=O].P.S.K.D.A.F.I.G.L.M.[am]}$$$$
PEPTIDE1{[N1[C@@H](CCC1=O)C(*)=O |$;;;;;;;;_R2;$|].P.S.K.D.A.F.I.G.L.M.[am]}$$$$
```

In case when the *OEHelmToMol* function successfully parses a HELM string the atoms that are generated for a specific monomer are added to a molecule as a group (*OEGroupBase*)

See also

- [Zhang-2012] publications
- *OEGetHelmMonomerCodes* function
- *OEGetHelmChainNames* function
- *OEMolToHelm* function

OpenEye Python Cookbook Code Example

- Convert HELM to SMILES
- Convert HELM File to Molecule File

OEIsPeptide

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEIsPeptide(const OEMolBase &, const OEMonomerSet& monomers,
                 double ratio = 0.8)
```

Determines whether the given molecule can be classified as a peptide with respect to the specified monomers. The function identifies backbone peptide monomers within the molecule to determine which atoms may belong to the peptide. If the ratio of matched atoms to total heavy atoms exceeds the specified threshold, the function returns true.

mol
The input molecule (*OEMolBase*).

monomers
The monomer set (*OEMonomerSet*) that stores the monomers that are used to interpret the HELM string.

ratio
Specifies the minimum *heavy atoms* to *matched monomer atoms* ratio required for a molecule to be considered a peptide. The ratio must fall within the range [0.25,0.90].

OEIsValidMonomerData

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEIsValidMonomerData(const OEMonomerData& monomer)
bool OEIsValidMonomerData(const OEMonomerData& monomer,
                          OEMonomerValidationResult& result)
```

Returns whether the *OEMonomerData* object is fully defined and correct.

See also

- *OEMonomerData* class
- *OEMonomerValidationResult* class and *OEMonomerValidationReturnCode* namespace

OELoadOpenEyeMonomerSet

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OELoadOpenEyeMonomerSet(OEMonomerSet& monomers)
```

Loads the built-in `OpenEye` monomer set containing more than 260 diverse monomers that are common in macro-cyclic peptide projects. This monomer set defines four code-sets:

- `OpenEye` - the primary code-set, the codes assigned to monomers follow naming conventions defined in [Ghiandoni-2026]
- `Standard` - 20 standard amino-acids with universally accepted one-letter codes, such as `C` for cystine
- `PDB` - monomer that also exists in PDB database with a three-letter code, such as `CYS` for cystine
- `ChEMBL` - monomer that also exists in the monomer dictionary of CHEMBL36.

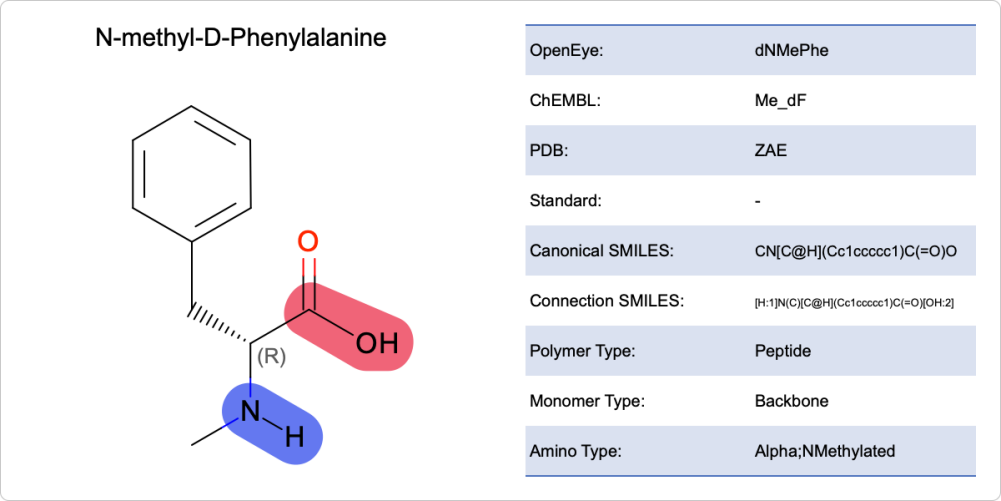


Fig. 16: **Example of monomer that occurs in different code-sets in the OpenEye monomer set**

Warning

The built-in `OpenEye` monomer set is intended primarily for testing, illustrating how OEChem handles diverse peptide structures during HELM parsing and generation, and providing a template for constructing private monomer sets.

Because this monomer set is **subject to change**, we strongly discourage our customers from relying on it in their production workflows. Monomers in this set may be added, removed, or modified between releases. Instead, users should define, validate, and maintain their own custom monomer set so they can control which monomers are included and adopt naming conventions that match their organizational standards.

See also

- *OEMonomerSet* class
- *OEReadMonomerSet* function
- *OELoadStandardMonomerSet* function

OELoadStandardMonomerSet

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OELoadStandardMonomerSet(OEMonomerSet& monomers)
```

Loads the built-in `Standard` monomer set containing 20 standard amino acids.

See also

- *OEMonomerSet* class
- *OEReadMonomerSet* function
- *OELoadOpenEyeMonomerSet* function

OEMonomerConnectionSmilesFromCXSmiles

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::string OEMonomerConnectionSmilesFromCXSmiles(const std::string& monomerCXSmiles,
                                                  const std::vector<std::string>&␣
↪cappingCXSmiles)
```

Converts CXSMILES style monomer definition for the OpenEye preferred style.. For example in case of alanine: `"[H:1]N[C@@H](C)C(=O)[OH:2]"` will be returned for the the `"C[C@H](N[*])C([*])=O |$;;;_R1;;_R2;$|"` cx-smiles and corresponding `"[*][H] |$_R1;$|"` and `"O[*] |$;_R2$|"` capping smiles. If the function fails an empty string will be returned.

Warning

The CXSMILES used in this function shouldn't contains any features other than the R-group number information.

See also

- *OECreatePeptideMonomerConnectionSmiles* function

OEMolToHelm

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::string OEMolToHelm(const OEMolBase& mol,
                        const OEMonomerSet& monomers)
std::string OEMolToHelm(const OEMolBase& mol,
                        const OEMonomerSet& monomers,
                        OEHelmGenerationResult& result)
std::string OEMolToHelm(const OEMolBase& mol,
                        const OEMonomerSet& monomers,
                        const OEHelmGenerationOptions& options)
std::string OEMolToHelm(const OEMolBase& mol,
                        const OEMonomerSet& monomers,
                        const OEHelmGenerationOptions& options,
                        OEHelmGenerationResult& result)
```

Converts a molecule into HELM string. The function will return HELM string if it can be generated successfully, otherwise the function will return an empty string.

mol
: The molecule (*OEMolBase*) the HELM string is generated from.

monomers
: The monomer set (*OEMonomerSet*) that stores the monomers that are used to generate the HELM string.

  If the given monomer set supports multiple code sets, *OEHelmToMol* automatically selects the primary code set.

options
: The options for HELM generation (*OEHelmGenerationOptions*).

result
: Is specified, the *OEHelmGenerationResult* object will store additional information about the success or failure of the HELM generation.

Example

See also

- [Zhang-2012] publications
- *OEHelmToMol* function

OpenEye Python Cookbook Code Example

- Convert SMILES to HELM
- Convert Molecule File to HELM File

OEMonomerTypeToString

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::string OEMonomerTypeToString(unsigned)
```

Converts a constant of the *OEMonomerType* namespace into string representation.

See also

- *OEMonomerType* namespace
- *OEStringToMonomerType* function

OEPerceivePeptideBackbone

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEPerceivePeptideBackbone(OEMolBase &)
```

The *OEPerceivePeptideBackbone* function identifies the peptide backbone of a molecule based on monomers that have already been perceived. It returns *true* if the backbone is successfully perceived and *false* if no monomer information is available. The function sets the following atom name constants *OEPDBAtomName.C*, *OEPDBAtomName.O*, *OEPDBAtomName.OXT*, *OEPDBAtomName.CA*, and *OEPDBAtomName.N* for atoms belonging to alpha amino acids. Furthermore, *OEPDBAtomName.CB*, *OEPDBAtomName.CG*, and *OEPDBAtomName.CD* are used to mark additional carbon atoms that are part of the backbone in beta, gamma, or delta peptide monomers.

mol
: The input molecule (*OEMolBase*).

Warning

The *OEPerceivePeptideBackbone* function does not perform a full residue perception like *OEPerceiveResidues* does. Instead, it removes any residue information (*OEResidue*) already present and marks only the backbone atoms based on monomer perception.

See also

- *OEGetPDBAtomIndex* function to access PDB ideas assigned to the atoms

Monomer information is required for the perception of the peptide backbone. The following functions will help in obtaining this information:

- *OEHelmToMol* function to convert a HELM string to a molecule
- *OEDetectMonomers* function to detect monomers in a molecule

OEPerceivePeptideMonomerConnections

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEPerceivePeptideMonomerConnections(OEMolBase& mol)
```

The function identifies the backbone of the amino acid represented by the molecule and marks the backbone connections in the molecule. The function returns whether both connection points could be identified.

See also

- *OECreatePeptideMonomerConnectionSmiles* function

OEPolymerTypeToString

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
std::string OEPolymerTypeToString(unsigned)
```

Converts a constant of the *OEPolymerType* namespace into string representation.

See also

- *OEPolymerType* namespace
- *OEStringToPolymerType* function

OEReadMonomerSet

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEReadMonomerSet(OEMonomerSet& monomers,
                      const std::string& jsonFileName)
```

Reads all monomer definitions from the given json file and adds them to the the *OEMonomerSet* objects.

```
bool OEReadMonomerSet(OEMonomerSet& monomers,
                      const std::string& jsonFileName,
                      const std::string& codeSet)
```

Reads all monomer definitions from the given json file and but only adds those to the *OEMonomerSet* objects which are defined in the given code-set.

See also

- *OEMonomerSet* class
- *OELoadStandardMonomerSet* and *OELoadOpenEyeMonomerSet* functions
- *OEWriteMonomerSet* function

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

OEStringToMonomerType

```
unsigned OEStringToMonomerType(const std::string &)
```

Converts a string into a constant in the *OEMonomerType* namespace. Returns *OEMonomerType.Undefined*, if the string does not represent any of the specific monomer type.

See also

- *OEMonomerType* namespace
- *OEMonomerTypeToString* function

OEStringToPolymerType

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
unsigned OEStringToPolymerType(const std::string &)
```

Converts a string into a constant in the *OEPolymerType* namespace. Returns *OEPolymerType.Undefined*, if the string does not represent any of the specific polymer types.

See also

- *OEPolymerType* namespace
- *OEPolymerTypeToString* function

OEWriteMonomerSet

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEWriteMonomerSet(const OEMonomerSet& monomers,
                       const std::string& jsonFileName)
```

Writes the monomer definitions stored in the monomer set (*OEMonomerSet*) into a JSON format.

The following example shows how a monomer is represented in the output file:

```
{
  "smiles": "[H:1]N(C)[C@@H](Cc1ccccc1)C(=O)[OH:2]",
  "name": "N-alpha-methyl-L-Phenylalanine",
  "codes": {
     "OpenEye": "MeF",
     "PDB": "MEA"
  }
},
```

The *OEPolymerType.Peptide* and *OEMonomerType.Backbone* are considered defaults for a monomer and are only written to the json file if their values differ from these defaults.

4.3 OESystem API

4.3.1 OESystem Classes

IsFalse

Warning

`IsFalse` is deprecated. Please use *OEIsFalse* instead.

```
template<class T> class IsFalse : public OEIsFalse<T>
```

The following methods are publicly inherited from *OEIsFalse*:

operator()	*CreateCopy*

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

IsTrue

> **Warning**
>
> `IsTrue` is deprecated. Please use *OEIsTrue* instead.

```
template<class T> class IsTrue : public OEIsTrue<T>
```

The following methods are publicly inherited from *OEIsTrue*:

operator()	*CreateCopy*

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

OEAnd

```
template<class Arg>
class OEAnd : public OEUnaryPredicate<Arg>
```

This class represents *OEAnd*.

This class template is a composition predicate which generates a predicate which is the logical *and* operation of the two predicates passed as arguments.

See also

- *Composite Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following specializations exist for this template:

OEAndAtom	OESystem::OEAnd<OEChem::OEAtomBase>
OEAndBond	OESystem::OEAnd<OEChem::OEBondBase>
OEAndConf	OESystem::OEAnd<OEChem::OEConfBase>
OEAndGroup	OESystem::OEAnd<OEChem::OEGroupBase>
OEAndRoleSet	OESystem::OEAnd<OEChem::OERoleSet>
OEAndInteractionHint	OESystem::OEAnd<OEChem::OEInteractionHint>

Constructors

```
OEAnd()
OEAnd(const OEAnd& rhs)
OEAnd(const OEUnaryPredicate<Arg>& func1,
      const OEUnaryPredicate<Arg>& func2)
```

Default and copy constructors.

operator()

```
bool operator()(const Arg& arg) const
```

CreateCopy

```
OEUnaryFunction<Arg, bool>* CreateCopy() const
```

OEAnnotation

```
class OEAnnotation : public OESystem::OEBase
```

This class represents *OEAnnotation*.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEAnnotation()
OEAnnotation(const OEAnnotation& rhs)
OEAnnotation(unsigned int type, const char* title = 0)
```

Default and copy constructors.

AddChild

```
void AddChild(const OEAnnotation& child)
```

CreateCopy

```
OEBase* CreateCopy() const
```

GetChildren

```
OESystem::OEIterBase<OEAnnotation>* GetChildren() const
```

GetColor

```
OESystem::OEColor GetColor() const
```

GetDataType

```
const void* GetDataType() const
```

GetRenderHint

```
unsigned int GetRenderHint() const
```

GetTitle

```
const char* GetTitle() const
```

GetType

```
unsigned int GetType() const
```

IsDataType

```
bool IsDataType(const void* type) const
```

SetColor

```
OEAnnotation& SetColor(const OESystem::OEColor &)
```

SetRenderHint

```
OEAnnotation& SetRenderHint(unsigned int hint)
```

SetTitle

```
void SetTitle(const char *)
```

SetType

```
void SetType(unsigned int type)
```

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEBInitializer

```
template<OEBHandInitializer INIT>
class OEBInitializer
```

Constructors

```
OEBInitializer()
```

Default constructor

OEBase

```
class OEBase
```

This class represents *OEBase*.

The abstract class *OEBase* defines the interface for run-time class extensibility and run-time type identification. Classes which derive from *OEBase* can store and retrieve data by association with integer or character string 'tag' identifiers.

All primitive data types can be stored through the mechanism provided by *OEBase*.

The following classes derive from this class:

- *OEAtomBase*
- *OEBondBase*
- *OEConfBase*
- *OEGraphMol*
- *OEMCMolBase*
- *OEMolBase*
- *OEQAtomBase*
- *OEQBondBase*
- *OEQMolBase*
- *OEAnnotation*
- *OEBinaryIOHandler*
- *OEBinaryIOHandlerBase*
- *OEContainer*
- *OEFixedGrid*
- *OEFixedMultiGrid*
- *OEGaussian*
- *OEGaussianBase*
- *OEGaussianContainer*
- *OEGaussianContainerBase*
- *OEGrid*
- *OEGridAbstractBase*
- *OEGridBase*
- *OEHeader*
- *OEMultiGrid*
- *OEScalarGrid*
- *OESkewGrid*

Constructors

```
OEBase()
OEBase(const OEBase& rhs)
```

Default and copy constructors.

operator+=

```
OEBase& operator+=(const OEBase &)
```

AddBaseData

```
bool AddBaseData(const OEBaseData *)
```

AddData

```
template<typename T>
bool AddData(const char* tag, const T t)
template<typename T>
bool AddData(unsigned int tag, const T t)
```

Stores and associates a copy of the data passed as the second argument with the 'tag' identifier given as the first argument. Integer tags should be allocated using the *OEGetTag* function. Multiple calls to *OEBase.AddData* of a *OEBase* derived class instance will result in multiple copies of data being stored. The *OEBase.AddData* method does not overwrite data stored by previous calls.

```
template<typename T>
bool AddData(const char* tag, const T t, unsigned int len)
template<typename T>
bool AddData(unsigned int tag, const T t, unsigned int len)
```

Stores and associates a copy of the data pointed to by the second argument with the 'tag' identifier given as the first argument. The third argument passed to the method is used to denote the length of the array pointed to by the second argument. The array length must be specified so that data can be copied directly. Integer tags should be allocated using the *OEGetTag* function. Multiple calls to *OEBase.AddData* of a *OEBase* derived class instance will result in multiple copies of data being stored. The *OEBase.AddData* method does not overwrite data stored by previous calls.

Clear

```
void Clear()
```

Clears all data stored using the *OEBase.AddData* and *OEBase.SetData* methods. When writing classes derived from *OEBase* the *OEBase.Clear* method should be chained to call the parent class *OEBase.Clear* method.

CreateCopy

```
OEBase* CreateCopy() const = 0
```

This pure virtual copy constructor should return a deep copy of the *OEBase* derived object. The copy of the object returned is dynamically allocated and should be deallocated using the C++ delete operator.

DeleteData

```
bool DeleteData(const char* tag)
bool DeleteData(unsigned int tag)
```

Deletes all instances of data stored with the *OEBase.AddData* and *OEBase.SetData* methods which are associated with the data 'tag' passed to the method. The method will return `true` if any data was deleted, and `false` if not data associated with the data tag was found.

GetBoolData

```
bool GetBoolData(const char* tag) const
bool GetBoolData(unsigned int tag) const
```

Returns the first instance of boolean data stored previously with the associated data tag. This method is equivalent to calling the *OEBase.GetData* method with template argument bool.

GetData

```
template<typename T>
const T& GetData(const char* tag) const
template<typename T>
const T& GetData(unsigned int tag) const
```

Returns the first instance of data stored previously with the associated data tag. Integer tags should be allocated using the *OEGetTag* function. Run-time type checking is performed when data is requested. A data type mismatch between the storage and request types will result in a default data instance being returned instead of the original data. Request for datum not present will also result in a default data instance being returned.

GetDataIter

```
OESystem::OEIterBase<OESystem::OEBaseData>* GetDataIter() const
```

Returns an iterator over data stored in an *OEBase* derived class instance. Data tag identification, run-time type checking, and data retrieval can be performed using the *OEBaseData* API.

See also

- *OEBaseDataT* template class

```
OESystem::OEIterBase<OESystem::OEBaseData>* GetDataIter(unsigned int tag) const
```

Returns an iterator over all generic data attached to the *OEBase* object with the given data *tag*.

See also

- *OEBaseDataT* template class

GetDataType

```
const void* GetDataType() const = 0
```

This pure virtual function is used to perform run-time type identification. The value returned by the method should be equivalent to the value returned by *OEGetDataType* using the derived class type as the template argument.

GetDoubleData

```
double GetDoubleData(const char* tag) const
double GetDoubleData(unsigned int tag) const
```

Returns the first instance of double precision floating point data stored previously with the associated data tag. This method is equivalent to calling the *OEBase.GetData* method with template argument `double`.

GetFloatData

```
float GetFloatData(const char* tag) const
float GetFloatData(unsigned int tag) const
```

Returns the first instance of single precision floating point data stored previously with the associated data tag. This method is equivalent to calling the *OEBase.GetData* method with template argument `float`.

GetIntData

```
int GetIntData(const char* tag) const
int GetIntData(unsigned int tag) const
```

Returns the first instance of integer data stored previously with the associated data tag. This method is equivalent to calling the *OEBase.GetData* method with template argument `int`.

GetStringData

```
const std::string& GetStringData(const char* tag) const
const std::string& GetStringData(unsigned int tag) const
```

Returns the first instance of an STL string stored previously with the associated data tag. This method is equivalent to calling the *OEBase.GetData* method with template argument `std::string`.

HasData

```
bool HasData(const char* tag) const
bool HasData(unsigned int tag) const
```

Returns `true` if any data stored in an *OEBase* derived object is associated with a particular data tag identifier. If no stored data is associated with the passed data 'tag' the methods return `false`.

IsDataType

```
bool IsDataType(const void *) const = 0
```

This method is used for run-time type identification of *OEBase* derived classes. The const void pointer argument passed to the function should be a value returned by the template function *OEGetDataType*. The *OEBase.IsDataType* method will return `true` if the data type identifier passed to the function matches at least one of the data types in the inheritance tree of the requested *OEBase* derived object. The method will return `false` if the data type identifier passed to the method fails to match any class type in the inheritance tree of the requested *OEBase* derived object. Implementations of this pure virtual method should chain to their parent class' *OEBase.IsDataType* method.

SetBaseData

```
bool SetBaseData(const OEBaseData *)
```

SetBoolData

```
bool SetBoolData(const char* tag, bool t)
bool SetBoolData(unsigned int tag, bool t)
```

Stores boolean values associated with the passed data 'tag' identifiers. The methods are equivalent to the *OEBase.SetData* method called with a boolean value.

SetData

```
template<typename T>
bool SetData(const char* tag, const T t)
template<typename T>
bool SetData(unsigned int tag, const T t)
```

Stores a copy of primitive or user-defined data passed as the second argument with a data 'tag' identifier passed as the first argument. If data of the same type has been stored using the identical tag in a previous *OEBase.AddData* or *OEBase.SetData* call, subsequent calls to *OEBase.SetData* will overwrite the first instance of data stored with the data tag. If a call to SetData succeeds in storing or overwriting previously stored data the method will return `true`. *OEBase.SetData* will return `false` if a copy of the data cannot be made. If data has been stored previously associated with a data tag, subsequent calls to *OEBase.SetData* with the identical data 'tag' but with different data type will result in a run-time warning being written and a return value of `false`. Note that these methods will not make copies of data passed by pointer. Values to be passed by pointer, or arrays, should be stored using the following *OEBase.SetData* methods.

```
template<typename T>
bool SetData(const char* tag, const T t, unsigned int len)
template<typename T>
bool SetData(unsigned int tag, const T t, unsigned int len)
```

Stores copies of arrays of primitive or user-defined data passed as the second argument with a data tag identifier passed as the first argument. The length of the array (number of elements) is passed as the third argument to the method. If data of the same type has been stored using the identical 'tag' in a previous *OEBase.AddData* or *OEBase.SetData* call, subsequent calls to *OEBase.SetData* will overwrite the first instance of data stored with the data tag. The array copy stored previously will be deallocated and a copy of the new array will be stored. If a call to *OEBase.SetData* succeeds in storing or overwriting an array stored previously the method will return `true`. *OEBase.SetData* will return `false` if a copy of the data cannot be made. If data has been stored previously associated with a data tag, subsequent calls to *OEBase.SetData* with the identical data 'tag' but with different data type will result in a run-time warning being written and a return value of `false`.

SetDoubleData

```
bool SetDoubleData(const char* tag, double t)
bool SetDoubleData(unsigned int tag, double t)
```

Stores and associates double precision floating point values passed as the second argument with the data tag provided as the first argument. The methods are provided for convenience and are functionally equivalent to calls to *OEBase.SetData* where the template argument is a double precision floating point number.

SetFloatData

```
bool SetFloatData(const char* tag, float t)
bool SetFloatData(unsigned int tag, float t)
```

Stores and associates single precision floating point values passed as the second argument with the data tag provided as the first argument. The methods are provided for convenience and are functionally equivalent to calls to *OEBase.SetData* where the template argument is a single precision floating point number.

SetIntData

```
bool SetIntData(const char* tag, int t)
bool SetIntData(unsigned int tag, int t)
```

Stores and associates integer values passed as the second argument with the data 'tag' provided as the first argument. The methods are provided for convenience and are functionally equivalent to calls to *OEBase.SetData* where the template argument is an integer value.

SetStringData

```
bool SetStringData(const char* tag, const std::string t)
bool SetStringData(unsigned int tag, const std::string t)
```

Stores and associates `std::string` passed as the second argument with the data 'tag' provided as the first argument. The methods are provided for convenience and are functionally equivalent to calls to *OEBase.SetData* where the template argument is a `std::string`.

OEBaseData

```
class OEBaseData
```

OEBaseData is an abstract class which defines the interface necessary for storage, retrieval, and duplication of data stored through the generic data interface in the *OEBase* class. Data stored in classes derived from *OEBaseData* is identified through data tags. The ability to perform run-time type checking is also built into the class.

The following classes derive from this class:

- *OEBaseDataT*
- *OEBaseDataTPtr*

Note

OEBaseData is an abstract class. Methods for accessing and setting generic data are implemented in its derived class *OEBaseDataT*.

See also

- *OEBaseDataT.GetData* method instead.
- *OEBaseDataT.SetData* method instead.

Constructors

```
OEBaseData(unsigned int tag)
```

Constructs and instance of *OEBaseData* and set the data identifier tag upon construction.

CreateCopy

```
OEBaseData* CreateCopy() const = 0
```

This pure virtual copy constructor should return a deep copy of the *OEBaseData* object. The copy of the object returned is not memory-managed by the parent instance. It is the responsibility of the programmer to manage the memory created and returned by this method.

GetDataType

```
const void* GetDataType() const = 0
```

This pure virtual method returns a unique type identifier for the data type being stored. The return value of the method is equivalent to the return value of *OEGetDataType* for the stored type.

GetSize

```
unsigned int GetSize() const = 0
```

This pure virtual method is used to return the number of elements if the data stored in an *OEBaseData* derived class is an array.

> **Note**
>
> The value returned is the number of elements in the array, not the number of bytes occupied by the data.

GetTag

```
unsigned int GetTag() const
```

Returns the integer data tag with which the data was stored.

OEBaseDataT

```
template<class T>
class OEBaseDataT : public OEBaseData
```

This class represents *OEBaseDataT*.

The following methods are publicly inherited from *OEBaseData*:

CreateCopy	*GetSize*
GetDataType	*GetTag*

The following specializations exist for this template:

- OESystem::OEBaseDataT< int >

The following classes derive from this class:

- *OEBaseDataTPtr*

Constructors

```
OEBaseDataT(unsigned int tag, const T& t)
```

Constructs an *OEBaseDataT* object that stores data with T type associated with the given `tag` identifier.

CreateCopy

```
OEBaseData* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEBaseDataT* object is dynamically allocated and owned by the caller.

GetData

```
T& GetData()
const T& GetData() const
```

Returns the data stored int the *OEBaseDataT* object.

GetDataType

```
const void* GetDataType() const
```

Returns a unique type identifier for the data type being stored. The return value of the method is equivalent to the return value of *OEGetDataType* for the stored type.

GetSize

```
unsigned int GetSize() const
```

Returns the number of elements stored in the an *OEBaseDataT* object that is always 1.

> **Note**
>
> The value returned is not the number of bytes occupied by the data.

SetData

```
bool SetData(const T& t)
```

Sets the data stored int the *OEBaseDataT* object.

OEBaseDataTPtr

```
template<class T>
class OEBaseDataTPtr : public OEBaseDataT<T>
```

This class represents *OEBaseDataTPtr*.

The following methods are publicly inherited from *OEBaseDataT*:

operator=	*GetData*	*GetSize*
CreateCopy	*GetDataType*	*SetData*

The following methods are publicly inherited from *OEBaseData*:

CreateCopy	*GetSize*
GetDataType	*GetTag*

The following specializations exist for this template:

- OESystem::OEBaseDataTPtr< OEBase * >
- OESystem::OEBaseDataTPtr< T * >

OEBinaryAnd

```
template<class Arg1, class Arg2>
class OEBinaryAnd : public OEBinaryPredicate<Arg1, Arg2>
```

This class represents *OEBinaryAnd*.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEBinaryAnd()
OEBinaryAnd(const OEBinaryAnd& rhs)
OEBinaryAnd(const OEBinaryPredicate<Arg1, Arg2>& func1,
            const OEBinaryPredicate<Arg1, Arg2>& func2)
```

Default and copy constructors.

operator()

```
bool operator()(const Arg1& arg1, const Arg2& arg2) const
```

CreateCopy

```
OEBinaryFunction<Arg1, Arg2, bool>* CreateCopy() const
```

OEBinaryFalse

```
template<class T, class U>
class OEBinaryFalse : public OEBinaryPredicate<T, U>
```

A functor to always return `false`.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const T &, const U &)
bool operator()(const T &, const U &) const
```

Always returns `false`.

CreateCopy

```
OESystem::OEBinaryFunction<T, U, bool>* CreateCopy() const
```

OEBinaryFunction

```
template<class Arg1, class Arg2, class Result, bool ConstArg, bool ConstFcn>
class OEBinaryFunction
```

This class represents *OEBinaryFunction*.

The following specializations exist for this template:

- OESystem::OEBinaryFunction< Arg1, Arg2, Result, false, false >
- OESystem::OEBinaryFunction< Arg1, Arg2, Result, false, true >
- OESystem::OEBinaryFunction< Arg1, Arg2, Result, true, false >

The following classes derive from this class:

- *OEInSamePart*

- *OEUnaryToBinaryAnd*
- *OEUnaryToBinaryOr*
- *OEBinaryAnd*
- *OEBinaryFalse*
- *OEBinaryNot*
- *OEBinaryOr*
- *OEBinaryPredicate*

operator()

```
Result operator()(const Arg1& a1, const Arg2& a2) const = 0
```

CreateCopy

```
base_type* CreateCopy() const = 0
```

OEBinaryIOHandler

```
class OEBinaryIOHandler : public OEBinaryIOHandlerBase
```

The following methods are publicly inherited form *OEBinaryIOHandlerBase*

Clear	*GetBaseWriteFxns*	*GetDefaultReadFxn*
GetDefaultWriteFxn	*GetReadFxns*	*GetWriteFxns*
ReplaceReadFxn	*ReplaceWriteFxn*	*SetBaseWriteFxn*
SetReadFxn	*SetWriteFxn*	

This class is for extremely advanced usage. Typical usage of this class is for preserving rotor offset compression. Please see the *OEPreserveRotCompress* function instead.

Constructors

```
OEBinaryIOHandler()
OEBinaryIOHandler(const OEBinaryIOHandler &)
OEBinaryIOHandler(const OEBinaryIOHandlerBase &)
```

Clear

```
void Clear()
```

CreateCopy

```
OEBase* CreateCopy() const
```

GetBaseWriteFxns

```
OEIterBase<OEWriteFxn<OEBaseData> >* GetBaseWriteFxns() const
```

GetDataType

```
const void* GetDataType() const
```

GetDefaultReadFxn

```
const OEReadFxnBase& GetDefaultReadFxn() const
```

GetDefaultWriteFxn

```
const OEWriteFxnBase& GetDefaultWriteFxn() const
```

GetReadFxns

```
OEIterBase<OEReadFxnBase>* GetReadFxns() const
```

GetWriteFxns

```
OEIterBase<OEWriteFxnBase>* GetWriteFxns() const
```

IsDataType

```
bool IsDataType(const void *) const
```

ReplaceReadFxn

```
bool ReplaceReadFxn (const OEReadFxnBase& ,
                     const std::string replacetag = std::string(""))
```

ReplaceWriteFxn

```
bool ReplaceWriteFxn(const OEWriteFxnBase&,
                     const std::string replacetag = std::string(""))
```

SetBaseWriteFxn

```
bool SetBaseWriteFxn(const OEWriteFxn<OEBaseData> &)
```

SetReadFxn

```
bool SetReadFxn(const OEReadFxnBase &, bool makedefault = false)
```

SetWriteFxn

```
bool SetWriteFxn(const OEWriteFxnBase &, bool makedefault = false)
```

OEBinaryIOHandlerBase

```
class OEBinaryIOHandlerBase : public OEBase
```

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

OEBinaryIOHandler

Clear

```
void Clear()=0
```

GetBaseWriteFxns

```
OEIterBase<OEWriteFxn<OEBaseData> >* GetBaseWriteFxns() const = 0
```

GetDefaultReadFxn

```
const OEReadFxnBase& GetDefaultReadFxn() const = 0
```

GetDefaultWriteFxn

```
const OEWriteFxnBase& GetDefaultWriteFxn() const = 0
```

GetReadFxns

```
OEIterBase<OEReadFxnBase>* GetReadFxns() const = 0
```

GetWriteFxns

```
OEIterBase<OEWriteFxnBase>* GetWriteFxns() const = 0
```

ReplaceReadFxn

```
bool ReplaceReadFxn (const OEReadFxnBase& ,
                     const std::string replacetag = std::string("")) = 0
```

ReplaceWriteFxn

```
bool ReplaceWriteFxn(const OEWriteFxnBase&,
                     const std::string replacetag = std::string("")) = 0
```

SetBaseWriteFxn

```
bool SetBaseWriteFxn(const OEWriteFxn<OEBaseData> &)=0
```

SetReadFxn

```
bool SetReadFxn(const OEReadFxnBase &, bool makedefault = false)=0
```

SetWriteFxn

```
bool SetWriteFxn(const OEWriteFxnBase &, bool makedefault = false)=0
```

OEBinaryNot

```
template<class Arg1, class Arg2>
class OEBinaryNot : public OEBinaryPredicate<Arg1, Arg2>
```

This class represents *OEBinaryNot*.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEBinaryNot()
OEBinaryNot(const OEBinaryNot<Arg1, Arg2>& rhs)
OEBinaryNot(const OEBinaryPredicate<Arg1, Arg2>& func)
```

Default and copy constructors.

operator()

```
bool operator()(const Arg1& a1, const Arg2& a2)
bool operator()(const Arg1& a1, const Arg2& a2) const
```

CreateCopy

```
OEBinaryFunction<Arg1, Arg2, bool>* CreateCopy() const
```

OEBinaryOr

```
template<class Arg1, class Arg2>
class OEBinaryOr : public OEBinaryPredicate<Arg1, Arg2>
```

This class represents *OEBinaryOr*.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEBinaryOr()
OEBinaryOr(const OEBinaryOr& rhs)
OEBinaryOr(const OEBinaryPredicate<Arg1, Arg2>& func1,
           const OEBinaryPredicate<Arg1, Arg2>& func2)
```

Default and copy constructors.

operator()

```
bool operator()(const Arg1& arg1, const Arg2& arg2) const
```

CreateCopy

```
OEBinaryFunction<Arg1, Arg2, bool>* CreateCopy() const
```

OEBinaryPredicate

```
template<class Arg1, class Arg2>
class OEBinaryPredicate : public OEBinaryFunction<Arg1, Arg2, bool>
```

This class represents *OEBinaryPredicate*.

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	CreateCopy

The following classes derive from this class:

- *OEInSamePart*
- *OEUnaryToBinaryAnd*
- *OEUnaryToBinaryOr*
- *OEBinaryAnd*
- *OEBinaryFalse*
- *OEBinaryNot*
- *OEBinaryOr*

operator()

```
bool operator()(const Arg1& a1, const Arg2& a2) const = 0
```

CreateCopy

```
base_type* CreateCopy() const = 0
```

CreatePredicateCopy

```
OEBinaryPredicate<Arg1, Arg2> *CreatePredicateCopy() const
```

OEBinaryTrue

```
template<class T, class U> class OEBinaryTrue : public OEBinaryPredicate<T, U>
```

A functor to always return `true`.

The following methods are publicly inherited from *OEBinaryPredicate*:

operator()	CreateCopy	CreatePredicateCopy

The following methods are publicly inherited from *OEBinaryFunction*:

operator()	CreateCopy

operator()

```
bool operator()(const T &, const U &)
bool operator()(const T &, const U &) const
```

Always returns `true`.

CreateCopy

```
OESystem::OEBinaryFunction<T, U, bool>* CreateCopy() const
```

OEBitVector

```
class OEBitVector
```

The *OEBitVector* class is used to represent a resizable bitmap. These are commonly used to store *fingerprints* in cheminformatics.

See also

- *OESubSearchScreen* class
- OEFingerPrint class in **GraphSim TK** manual

Constructors

```
OEBitVector()
OEBitVector(unsigned int size)
OEBitVector(const OEBitVector& src)
```

When constructed with a `size` argument, this specifies the initial number of bits in the *OEBitVector*. When constructed without an argument, the default number of bits is architecture dependent, `32` bits on 32-bit hosts and `64` bits on 64-bit hosts. Initially, all the bits are set to zero.

```
OEBitVector(OERandom& rand, unsigned int size)
```

Creates a *OEBitVector* with `size` bits randomly initialized from the `rand` object.

```
OEBitVector(const unsigned char* data, unsigned int size)
```

Using this constructor is equivalent to creating a default constructed *OEBitVector* and then immediately calling *OEBitVector.SetData* with the `data` and `size` arguments.

ClearBits

```
void ClearBits()
```

Clears all of the bits of the *OEBitVector* to zero.

CountBits

```
unsigned int CountBits() const
```

Counts the number of bits that are set to one, in the *OEBitVector* object.

CountRangeBits

```
unsigned int CountRangeBits(const unsigned start,
                            const unsigned end) const
```

Counts the number of bits that are set to one in the given bit range.

FirstBit

```
int FirstBit() const
```

Returns the index of the first bit, *i.e.* the set bit with the lowest index, in the *OEBitVector*. Bit position indices are numbered from zero, and have a maximum of *OEBitVector.GetSize* -1. If the *OEBitVector* has no bits set, *i.e.* *OEBitVector.IsEmpty* returns `true`, this method return will return the value -1.

FromHexString

```
void FromHexString(const char* bvs)
void FromHexString(const std::string& bvs)
```

Converts a hexadecimal string, in either upper, lower or mixed case, into an *OEBitVector*. Each valid hexadecimal digit (0-9, a-f, A-F) translates to four bits in the *OEBitVector*. Note: Because *OEBitVectors* may have a length not divisible by four, the last character of an OEChem-generated hexadecimal string may be a non-hex digit encoding the correct length of the *OEBitVector*. Don't worry if you see a non-hex digit at the end of a hexadecimal string.

See also

- *OEBitVector.ToHexString* method

GetData

```
const unsigned char* GetData() const
```

Returns a pointer to the internal storage of the *OEBitVector*'s bitmap. This pointer to a sequence of at least (*OEBitVector.GetSize* +7)/8 consecutive bytes. Bit index zero corresponds to the least significant bit of the first byte. All bits beyond *OEBitVector.GetSize* in the last byte are guaranteed to be zero. The *OEBitVector.GetData* method always returns a valid non-NULL pointer even if the number of bits is zero.

GetSize

```
unsigned int GetSize() const
```

Returns the size, in bits, of the *OEBitVector*. An *OEBitVector* may potentially contain zero bits.

IsBitOn

```
bool IsBitOn(unsigned int bit) const
```

Tests whether the bit as the specified position/index of an *OEBitVector* is set. Bit position indices are numbered from zero, and have a maximum of *OEBitVector.GetSize* -1. This method returns `false` for all indices greater than or equal to *OEBitVector.GetSize*.

IsEmpty

```
bool IsEmpty() const
```

Returns `true` if all the bits of an *OEBitVector* are zero. This is equivalent to, but much more efficient than, testing that *OEBitVector.CountBits* == 0. An *OEBitVector* of zero size is considered empty.

LastBit

```
int LastBit() const
```

Returns the index of the last bit, *i.e.* the set bit with the highest index, in the *OEBitVector*. Bit position indices are numbered from zero, and have a maximum of *OEBitVector.GetSize* -1. If the *OEBitVector* has no bits set, *i.e.* *OEBitVector.IsEmpty* returns `true`, this method return will return the value -1.

NegateBits

```
void NegateBits()
```

Inverts all of the bits of an *OEBitVector*.

NextBit

```
int NextBit(unsigned int) const
```

Returns the next set bit after the specified bit position, *i.e.* the set bit with the lowest index greater than the argument. If there are no such set bits, or the value of bit is greater than or equal to *OEBitVector.GetSize*, this method returns the value -1.

PrevBit

```
int PrevBit(unsigned int) const
```

Returns the previous set bit before the specified bit position, *i.e.* the set bit with the highest index less than the argument. If there are no such set bits, this method returns the value -1. If the specified value of bit is greater than or equal to *OEBitVector.GetSize*, this method returns the same values as *OEBitVector.LastBit*.

SetBitOff

```
void SetBitOff(unsigned int bit)
```

Clears the bit at the specified bit position/index of an *OEBitVector*. Bit position indices are numbered from zero, and have a maximum of *OEBitVector.GetSize* -1. If the value of 'bit' is greater than or equal to *OEBitVector.GetSize*, the bitmap is resized to 'bit'+1 bits. All of the new bits are initialized to zero.

SetBitOn

```
void SetBitOn(unsigned int bit)
```

Tests whether the bit as the specified position/index of an *OEBitVector* is set. Bit position indices are numbered from zero, and have a maximum of *OEBitVector.GetSize* -1. This method returns `false` for all indices greater than or equal to *OEBitVector.GetSize*.

SetData

```
void SetData(const unsigned char* data, unsigned int nbits)
```

Initialize the *OEBitVector* by the binary data pointed to by `data`, possibly resizing the *OEBitVector* to accommodate the `nbits` of data. `data` should point to at least `nbits/8` bytes of data. If a non-multiple of 8 bits is to be copied it is the users responsibility to make sure the remainder of bits in the last byte are zeroed out.

SetRangeOff

```
void SetRangeOff(unsigned int start, unsigned int end)
```

SetRangeOn

```
void SetRangeOn(unsigned int start, unsigned int end)
```

SetSize

```
void SetSize(unsigned int bits)
```

Resizes an *OEBitVector* to the specified number of bits. A size argument of zero is allowed. If this method increases the size of an *OEBitVector* all of the new bit positions are initialized to zero.

ToHexString

```
void ToHexString(std::string& bvs) const
```

Generates a hexadecimal string representation of an *OEBitVector*. Each "nibble" of four bits is converted into an uppercase hexadecimal character.

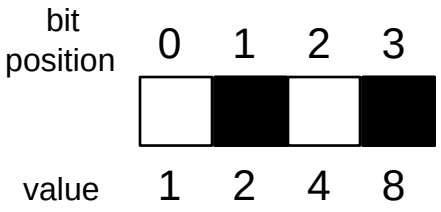


Fig. 17: **The interpretation of the four-bit long "nibble"-s (big-endian system)**

The four-bit long "nibble" shown above is encoded as hexadecimal 'A':

$0 * 2^0 + 1 * 2^1 + 0 * 2^2 + 1 * 2^3 = 0 + 2 + 0 + 8 = 10(dec) = A(hex)$

Because *OEBitVectors* may have a length that is not a multiple of four, an extra non-hexadecimal digit is added to the end of the string to encode the correct *OEBitVector* length.

Table 45: **Interpretation of last character**

last character	number of extra bits
'J'	0
'G'	1
'H'	2
'I'	3

For example, if the size of the bit-vector is 8 (multiple of four), then the last character of the returned string will be 'J'. If the size of the bit-vector is 9, then the last character will be 'I', since there will be three extra bits to add up to a number that can be multiplied by four.

See also

- *OEBitVector.FromHexString* method

ToggleBit

```
void ToggleBit(unsigned int bit)
```

Inverts the bit at the specified bit position/index of an *OEBitVector*. Bit position indices are numbered from zero, and have a maximum of *OEBitVector.GetSize* -1. If the value of 'bit' is greater than or equal to *OEBitVector.GetSize*, the bitmap is resized to 'bit'+1 bits. All of the new bits other than the one specified in the argument, *i.e.* the last, are initialized to zero.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEBoundedBuffer

```
template<class T>
class OEBoundedBuffer
```

First-in-first-out (FIFO) queue that only allows for `maxsize` items to be stored in the buffer, hence, "bounded". It is not inherently thread safe by itself. Use this in conjunction with a *OEProtectedBuffer* to make a thread-safe queue.

Constructors

```
OEBoundedBuffer(oesize_t maxsize)
```

Determines the maximum number of elements this queue can hold. The `maxsize` parameter should be a power-of-2 to allow for a more efficient implementation.

Empty

```
bool Empty() const
```

Returns whether the queue contains any items.

Full

```
bool Full() const
```

Returns `true` if the queue contains `maxsize` items.

Get

```
T Get()
```

Pops the next item off of the queue.

Peek

```
T Peek()
```

Returns the item that will be returned by the next call to *OEBoundedBuffer.Get*, but does not remove it from the queue.

Put

```
void Put(T obj)
```

Pushes an item into the queue.

Size

```
oesize_t Size() const
```

Returns the current number of items in the queue.

OEColor

```
class OEColor
```

The *OEColor* class represents colors in terms of RGBA (red, green, blue, alpha) components, where alpha is used as an opacity channel. Each color component is represented by one byte in the range of `0` to `255`. Meaning that $2^8 * 2^8 * 2^8 = 16,777,216$ different colors can be represented.

The value of alpha is also in the range of `0` to `255`. If a color has an alpha value of `0`, then the color is fully transparent, *i.e.*, invisible. If a color has an alpha value `255`, then the color is fully opaque.

OESystem provides many predefined colors. For example, the color black is represented by *OEBlack* (R=0, G=0, B=0), and the color white is represented by *OEWhite* (R=255, G=255, B=255). The full list of pre-defined colors is shown in *Figure: List of pre-defined colors*

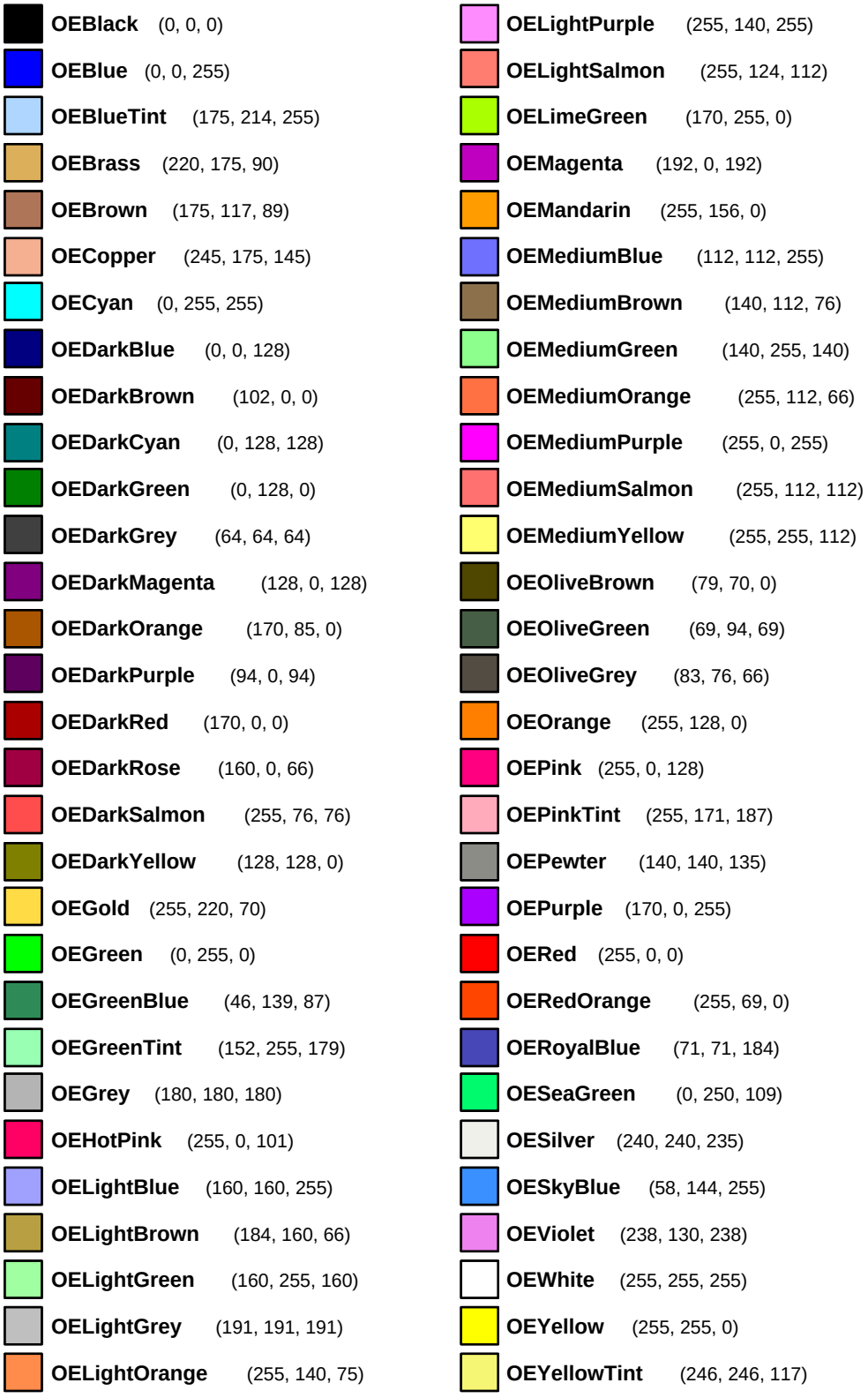


Fig. 18: **List of pre-defined colors**

> **Note**
>
> The alpha for all pre-defined colors is `255`, entirely opaque.

Constructors

```
OEColor()
```

The default constructor initializes the color object to *white* (R=255, B=255, G=255, A=255).

```
OEColor(const OEColor& rhs)
```

Copy constructor.

```
OEColor(unsigned int packed)
```

Create a color object from a packed 32-bit integer. Each 8-bit byte of the integer is assumed to contain a different component of the color.

```
OEColor(const std::string& text)
```

Create a color object that is initialized with an RGB or an RGBA hexadecimal string. The format of an RGB string is `#RRGGBB` and format of an RGBA string is `#RRGGBBAA`, where R,G,B and A are hexadecimal numbers. For example, the string `#FF0000` represents the color red.

```
OEColor(const unsigned char* rgba)
```

Create a color object from an array of `unsigned char`. The array is assumed to contain at least 4 bytes, where each byte corresponds to RGBA components respectively.

```
OEColor(unsigned int red, unsigned int green, unsigned int blue)
```

Create a color object with a specified *red*, *green*, and *blue* color components, the alpha component is set to `255`. Each argument should be in the range of `0` to `255`.

```
OEColor(unsigned int red, unsigned int green, unsigned int blue,
        unsigned int alpha)
```

Create a color object with a specified *red*, *green*, *blue*, and *alpha* components.

CreateCopy

```
OEColor* CreateCopy() const
```

Deep copy constructor that returns a pointer to a copy of the object. The memory for the returned *OEColor* object is dynamically allocated and owned by the caller.

GetR

```
unsigned int GetR() const
```

Returns the red component of the *OEColor* object in the range of `[0, 255]`.

See also

- *OEColor.SetR* method

GetG

```
unsigned int GetG() const
```

Returns the green component of the *OEColor* object in the range of `[0, 255]`.

See also

- *OEColor.SetG* method

GetB

```
unsigned int GetB() const
```

Returns the blue component of the *OEColor* object in the range of `[0, 255]`.

See also

- *OEColor.SetB* method

GetA

```
unsigned int GetA() const
```

Returns the alpha component of the *OEColor* object in the range of `[0, 255]`.

See also

- *OEColor.SetA* method

GetPackedColor

```
unsigned int GetPackedColor() const
```

Returns all the components of the *OEColor* object packed into a single 32-bit integer. Each 8-bit byte of the integer contains a different component of the color.

GetRGBA

```
const unsigned char* GetRGBA() const
```

Returns all the components of the *OEColor* object as an array of 4 bytes. The array contains 4 bytes, where each byte corresponds to RGBA components respectively. The pointer returned is owned by the *OEColor* object and should not be deallocated by the user.

GetText

```
std::string GetText(bool alpha = false) const
```

Returns the hexadecimal string representation of the *OEColor* object. By default the format is `#RRGGBB`. If the `alpha` argument is set to `true` the returned string will include the alpha component in the format `#RRGGBBAA`.

Set

```
OEColor& Set(unsigned int r, unsigned int g, unsigned int b)
OEColor& Set(unsigned int r, unsigned int g, unsigned int b,
             unsigned int a)
```

Sets all the components of the *OEColor* object. Each argument is assumed to be in the range of `0` to `255`. Returns a reference to the *OEColor* object.

SetR

```
OEColor& SetR(unsigned int r)
```

Sets the red component of the *OEColor* object. The given value is clamped into the range of `[0, 255]`. Returns a reference to the *OEColor* object.

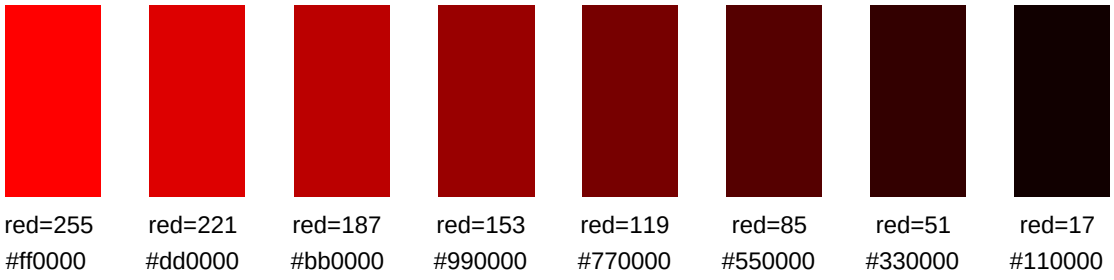


Fig. 19: **Examples of changing the red color component**

See also

- *OEColor.GetR* method

SetG

```
OEColor& SetG(unsigned int g)
```

Sets the green component of the *OEColor* object. The given value is clamped into the range of `[0, 255]`. Returns a reference to the *OEColor* object.

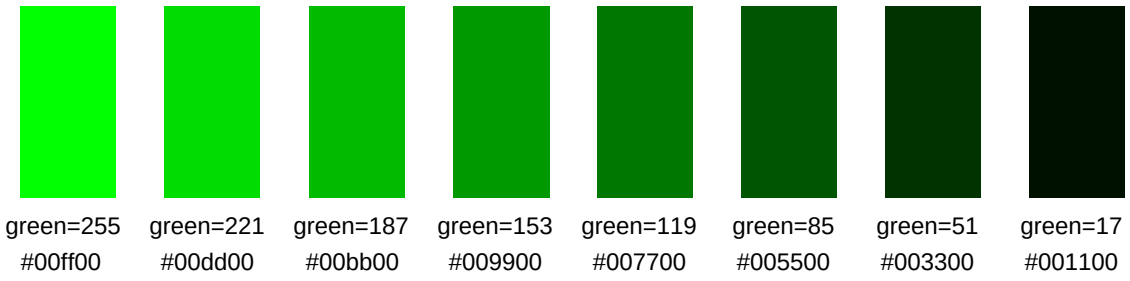


Fig. 20: **Examples of changing the green color component**

See also

- *OEColor.SetG* method

SetB

```
OEColor& SetB(unsigned int b)
```

Sets the blue component of the *OEColor* object. The given value is clamped into the range of `[0, 255]`. Returns a reference to the *OEColor* object.

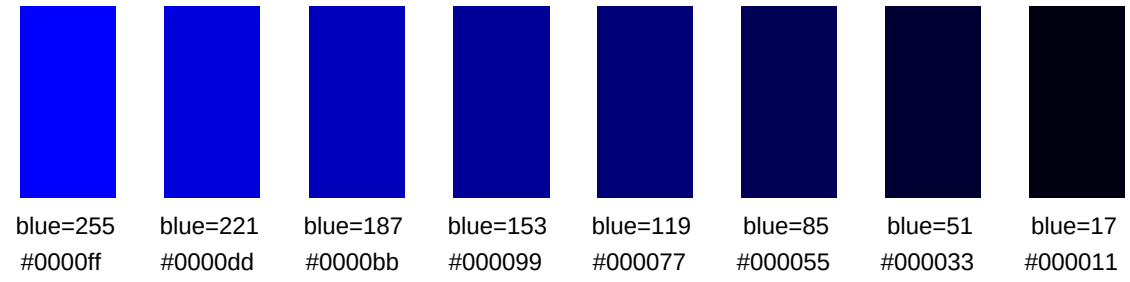


Fig. 21: **Examples of changing the blue color component**

See also

- *OEColor.SetB* method

SetA

```
OEColor& SetA(unsigned int a)
```

Sets the alpha component of the *OEColor* object. The given value is clamped into the range of `[0, 255]`. Returns a reference to the *OEColor* object.

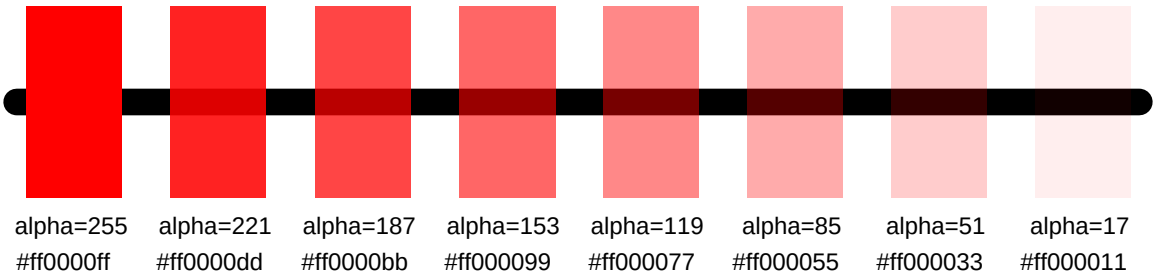


Fig. 22: **Examples of changing the alpha (transparency) color component**

See also

- *OEColor.SetA* method

SetPackedColor

```
OEColor& SetPackedColor(unsigned int color)
```

Sets all the components of the *OEColor* object from a packed 32-bit integer. Each 8-bit byte of the integer is assumed to contain a different component of the color. Returns a reference to the *OEColor* object.

SetRGBA

```
OEColor& SetRGBA(const unsigned char* rgba)
```

Sets all the components of the *OEColor* object from an array of `unsigned char`. The array is assumed to contain at least 4 bytes, where each byte corresponds to RGBA components respectively. Returns a reference to the *OEColor* object.

SetText

```
OEColor& SetText(const std::string& text)
```

Sets all the components of the *OEColor* object from a RGB or RGBA hexadecimal string. The format of an RGB string is `#RRGGBB` and format of an RGBA string is `#RRGGBBAA`, where R,G,B and A are hexadecimal numbers. For example, the string `#FF0000` represents the color red. Returns a reference to the *OEColor* object.

OEColorGradientBase

```
class OEColorGradientBase
```

OEColorGradientBase is an abstract class used for interpolating between colors that are defined by using the *OEColorStop* class.

The following classes derive from this class:

- *OEExponentColorGradient*
- *OEExponentialColorGradient*

- *OELinearColorGradient*
- *OELogarithmicColorGradient*

Constructors

```
OEColorGradientBase()
```

Default constructor that creates an "empty" color gradient.

> **Note**
>
> An "empty" color gradient always returns the *OEBlack* color, when calling the *OEColorGradientBase.GetColorAt*, the *OEColorGradientBase.GetMinColor* or the *OEColorGradientBase.GetMaxColor* methods.

```
OEColorGradientBase(const OEColorStop& oneStop)
```

Creates an *OEColorGradientBase* object with one color stop. An *OEColorGradientBase* object that has only one color stop will always return that color when calling the *OEColorGradientBase.GetColorAt* method.

```
OEColorGradientBase(const OEColorStop& bgnStop,
                    const OEColorStop& endStop)
```

Creates an *OEColorGradientBase* object with interpolation range between the two specified color stops.

```
OEColorGradientBase(const OEColorGradientBase& rhs)
```

Copy constructor.

AddStop

```
bool AddStop(const OEColorStop& stop)
```

Add a new color stop to the color gradient.

DeleteStop

```
bool DeleteStop(unsigned int stopidx)
```

Deletes the color stop at `idx` and returns `true`. If the color gradient only has one color stop or the `idx` is not between `0` and *NumStops()*, then *OEColorGradientBase.DeleteStop* throws a warning and returns `false`.

GetColorAt

```
OEColor GetColorAt(double value) const
```

Returns the interpolated color at `value` defined by the color stops in the color gradient. Returns *OEColorGradientBase.GetMinColor* if the `value` is less than *OEColorGradientBase.GetMinValue*. Returns *OEColorGradientBase.GetMaxColor* if the `value` is greater than *OEColorGradientBase.GetMinValue*.

GetMaxColor

```
const OEColor& GetMaxColor() const
```

Returns the color associated with the value returned by the *OEColorGradientBase.GetMaxValue* method.

GetMaxValue

```
double GetMaxValue() const
```

Returns the highest value stored in any of the color stops of the color gradient.

GetMinColor

```
const OEColor& GetMinColor() const
```

Returns the color associated with the value returned by the *OEColorGradientBase.GetMinValue* method.

GetMinValue

```
double GetMinValue() const
```

Returns the lowest value stored in any of the color stops of the color gradient.

GetStops

```
OEIterBase< OEColorStop >* GetStops() const
```

Returns an iterator over all color stops of the color gradient. The iterator is invariant to further modifications of the *OEColorGradientBase* object.

IsEmpty

```
bool IsEmpty() const
```

Returns `true` if no color stop has been added to the *OEColorGradientBase* object that has been created by the default constructor, otherwise it returns `false`.

NumStops

```
unsigned int NumStops() const
```

Returns the number of color stops of the color gradient.

TransformValue

```
double TransformValue(double value) const = 0
```

This is a virtual const method that is implemented in the concrete color gradient derived classes. This method is called by *OEColorGradientBase.GetColorAt* method to transform a value before performing color interpolation.

OEColorStop

```
class OEColorStop
```

The *OEColorStop* class stores a color (*OEColor*) and value (double) pair that is used by the color gradient classes.

See also

- *OEColorGradientBase* base class
- *OEExponentColorGradient* class
- *OEExponentialColorGradient* class
- *OELinearColorGradient* class
- *OELogarithmicColorGradient* class

Constructors

```
OEColorStop(double value, const OEColor& color)
```

Default constructor.

```
OEColorStop(const OEColorStop& rhs)
```

Copy constructor.

GetColor

```
const OEColor& GetColor() const
```

Returns the *OEColor* stored in the *OEColorStop* object.

GetValue

```
double GetValue() const
```

Returns the value stored in the *OEColorStop* object.

Warning

This functionality is only available in the C++ toolkits.

OEConcatIter

```
template<class A>
class OEConcatIter : public OEIterBase<A>
```

This class represents *OEConcatIter*.

The following methods are publicly inherited from *OEIterBase*:

*operator**	*operator-=*	*Copy*
operator++	*operator->*	*Push*
operator+=	*operator A **	*ToFirst*
operator--	*operator bool*	*ToLast*

Constructors

```
OEConcatIter()
OEConcatIter(OEIterBase<A>* rhs)
OEConcatIter(const OEConcatIter<A>& rhs)
OEConcatIter(OEIterBase<A>* ib1, OEIterBase<A>* ib2)
```

Default and copy constructors.

operator bool

```
operator bool() const
```

Copy

```
OEIterBase<A> *Copy() const
```

Push

```
OEIterBase<A> *Push(OEIterBase<A> *ib)
```

ToFirst

```
OEIterBase<A> &ToFirst()
```

ToLast

```
OEIterBase<A> &ToLast()
```

OEConsoleProgressTracer

```
class OEConsoleProgressTracer : public OETracerBase
```

The following methods are publicly inherited from *OETracerBase*:

Aborted	*Push*	*SetProgress*
Pop	*Refresh*	*SetTask*

Constructors

```
OEConsoleProgressTracer()
```

Default constructor.

Aborted

```
bool Aborted()
```

This is a no-op method for this class.

Pop

```
void Pop()
```

Prints out the task name and the progress bar along with the elapsed time. An example output is shown below.

```
Screen generation       [========------------------------------------------------------] ␣
↪13.53%  00:00:53
```

Push

```
void Push(std::string task)
```

This is equivalent to calling *OEConsoleProgressTracer.SetTask*.

Refresh

```
void Refresh()
```

This is a no-op method for this class.

SetProgress

```
void SetProgress(double percent)
void SetProgress(unsigned int current, unsigned int total)
```

Prints out the task name and its progress bar when the current progress of the task has reached the percentage of completion specified by `percent` or by `current` / `total`.

SetTask

```
void SetTask(std::string task)
```

Registers the tracer with a task name specified by, `task`. It also starts the timer to trace the progress of the task and prints out the initial progress bar, i.e. `0.0`.

OEContainer

```
class OEContainer : public OEBase
```

This class represents *OEContainer*.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEHeader*

Constructors

```
OEContainer()
OEContainer(const OEBase& rhs)
OEContainer(const OEContainer& rhs)
```

Default and copy constructors.

CreateCopy

```
OEBase* CreateCopy() const
```

GetDataType

```
const void* GetDataType() const
```

IsDataType

```
bool IsDataType(const void* type) const
```

OECycleTimer

```
class OECycleTimer
```

This class represents *OECycleTimer*.

Constructors

```
OECycleTimer()
```

Default and copy constructors.

Elapsed

```
OEULongLong Elapsed()
```

Lap

```
OEULongLong Lap()
```

Start

```
void Start()
```

OEDots

```
class OEDots
```

Provides a simple progress bar interface for command line applications. Dots, `.`, characters are printed to *OEPlatform.oeerr* to indicate to the user of the command line application that progress is indeed being made and the program is not hung indefinitely.

See also

A version of this class that is safe to used from multiple threads of computation: *OEThreadedDots*.

Constructors

```
OEDots(unsigned int bigstep, unsigned int smallstep,
       const char * objectname = "",
       const char * action = "processed")
```

Controls how the following output gets updated:

```
.......... 10000 things processed.
.......... 20000 things processed.
.......... 30000 things processed.
.......... 40000 things processed.
.......... 50000 things processed.
.......... 60000 things processed.
.......... 70000 things processed.
.......... 80000 things processed.
```

bigstep
The number of elements before a new line is generated. The above example is a `bigstep` of `10000`.

smallstep
The number of elements before a single new dot, `.`, is printed. The above example is a `smallstep` of `1000`. Or 10 dots per line.

objectname
A description of what is being processed. In the above example, this is “things”.

action
A description of what is done to the 'objects'. In the above example, this is "processed".

GetBigStep

```
unsigned int GetBigStep()
```

Returns the `bigstep` size used to construct this instance.

GetCounts

```
unsigned int GetCounts()
```

Returns the current number of elements counted by this instance.

GetSmallStep

```
unsigned int GetSmallStep()
```

Returns the `smallstep` size used to construct this instance.

SetBigStep

```
void SetBigStep(unsigned int s)
```

Sets a new `bigstep` size for this instance.

SetSmallStep

```
void SetSmallStep(unsigned int s)
```

Sets a new `smallstep` size for this instance.

Total

```
void Total()
```

Prints a final summary line like the following:

```
Total: 80000 things processed.
```

Typically only run once at the end of all processing.

Update

```
void Update(unsigned int step = 1)
void Update(const char symbol, unsigned int step = 1)
```

Update the progress with the number of elements specified by `step`. The default is 1 element, but any number can be specified. However, `0` is not a valid step size and will throw an error. The second overload allows to specify a *symbol* to be displayed as progress, instead of a dot.

Note

Not thread-safe, see *OEThreadedDots* if this method needs to be called from multiple threads.

See also

- Example program *Dots.cs*

OEErrorHandler

```
class OEErrorHandler
```

This class represents *OEErrorHandler*.

The *OEErrorHandler* class is used to define the behavior of the *OEThrow* error handler. All diagnostic output, including error, warning and informational messages, generated by the *OpenEye* libraries are emitted via *OEThrow*. The *OEErrorHandler* class defines how these messages are emitted, but also how the user can redirect and/or intercept these messages.

Constructors

```
OEErrorHandler()
```

Default constructor that creates an *OEErrorHandlerImplBase* that it owns and writes output to stderr.

```
OEErrorHandler(OEErrorHandler& rhs)
```

Copy constructor.

Note

OEErrorHandlerImplBase.CreateCopy will only be called by the copy constructor if the right-hand-side owns the *OEErrorHandlerImplBase*. This is why the right-hand-side argument is non-const.

Clear

```
void Clear()
```

Reset all the counts returned by *OEErrorHandler.Count* to zero.

Count

```
unsigned int Count(unsigned int level = 0) const
```

Returns the number of messages already written at the given `level`. The constant `level` should be drawn from the *OEErrorLevel* namespace. If `level` is zero, the default, the sum of all error level counts is returned.

Debug

```
void Debug(const std::string &)
void Debug(const char* format, ...)
```

Writes debugging messages to the *OEErrorHandler*. By default, these message will not appear, *OEErrorHandler.SetLevel* should be set to *OEErrorLevel.Debug* or below to make them appear.

Error

```
void Error(const std::string &)
void Error(const char* format, ...)
```

Writes error messages to the *OEErrorHandler*. Depending upon the value of the *OEErrorHandler*'s 'strict' property (see *OEErrorHandler.GetStrict* and *OEErrorHandler.SetStrict*), these diagnostics are forwarded to either the *OEErrorHandler.Fatal* or *OEErrorHandler.Warning* methods.

Fatal

```
void Fatal(const std::string &)
void Fatal(const char* format, ...)
```

Writes fatal error messages to the *OEErrorHandler*.

GetHandlerImpl

```
OEErrorHandlerImplBase* GetHandlerImpl()
```

Returns the *OEErrorHandlerImplBase* error handler that is used to handle messages. The default error handler writes output to stderr. The error handler can be changed by calling the *OEErrorHandler.SetHandlerImpl* method.

GetLevel

```
unsigned int GetLevel() const
```

Returns a constant from the *OEErrorLevel* namespace indicating the current level of messages that should make it to the output stream. The default is *OEErrorLevel.Default*, that will output messages of *OEErrorHandler.Info* or higher.

> **Warning**
>
> All threads, including the main thread, will start out at the *OEErrorLevel.Default* level. *OEErrorHandler.SetLevel* will only change the level for the current thread.

GetStrict

```
bool GetStrict() const
```

Returns the 'strict' property of the *OEErrorHandler*. When the *OEErrorHandler* is strict, all error messages written via the *OEErrorHandler.Error* methods are treated as fatal, and handled like those passed to *OEErrorHandler.Fatal*. When the *OEErrorHandler* is not strict, error messages written via *OEErrorHandler.Error* are treated like those

to *OEErrorHandler.Warning*. The strict property of an *OEErrorHandler* can be set using the *OEErrorHandler.SetStrict* method.

Info

```
void Info(const std::string &)
void Info(const char* format, ...)
```

Writes an informational messages to the *OEErrorHandler*.

Msg

```
void Msg(unsigned int level, const std::string &)
void Msg(unsigned int level, const char* format, ...)
```

Write a message at any `level`. The `level` constant should be drawn from the *OEErrorLevel* namespace.

SetHandlerImpl

```
void SetHandlerImpl(OEErrorHandlerImplBase* ptr, bool owned)
```

Use a different implementation of *OEErrorHandlerImplBase* to handle error messages. The ownership is determined by the `owned` flag, which is always `false` in any other language but C++.

This method allows a user to completely redefine how the **OpenEye toolkits** handle error messages by declaring a new subclass of *OEErrorHandlerImplBase* and defining the *OEErrorHandlerImplBase.Msg* to have the desired behavior.

SetLevel

```
void SetLevel(unsigned int errLevel)
```

Controls whether a message passed to the error handler will make it to the output stream. Values greater than `errLevel` from the *OEErrorLevel* namespace will be sent to the output stream.

> **Warning**
>
> This method will only change the error level for the current thread. Child threads will not inherit the parent thread error level.

SetOutputStream

```
void SetOutputStream(OEPlatform::oeostream& ofs)
```

Sets the output stream (*oeostream*) that diagnostic messages are written to when no custom error handler has been specified. By default, all messages are written to *OEPlatform.oeerr*. This function can be used to redirect this output to either a file stream (*oeofstream*) or a string stream (*oeosstream*).

SetStrict

```
void SetStrict(bool strict)
```

Sets the 'strict' property of the *OEErrorHandler*. When the *OEErrorHandler* is strict, all error messages written via the *OEErrorHandler.Error* methods are treated as fatal, and handled like those passed to *OEErrorHandler.Fatal*. When the *OEErrorHandler* is not strict, error messages written via *OEErrorHandler.Error* are treated like those to *OEErrorHandler.Warning*. The strict property of an *OEErrorHandler* can be retrieved by using the *OEErrorHandler.GetStrict* method.

Usage

```
void Usage(const std::string &)
void Usage(const char* format, ...)
```

Writes usage (warning) messages to the *OEErrorHandler*.

Verbose

```
void Verbose(const std::string &)
void Verbose(const char* format, ...)
```

Writes messages with the *OEErrorLevel.Verbose* error message level to the *OEErrorHandler*. By default, these messages will not appear.

Warning

```
void Warning(const std::string &)
void Warning(const char* format, ...)
```

Writes warning messages to the *OEErrorHandler*.

OEErrorHandlerImplBase

```
class OEErrorHandlerImplBase
```

This class represents *OEErrorHandlerImplBase*.

The *OEErrorHandlerImplBase* abstract base class is used to define the API that can be used to customize an *OEErrorHandler*, typically the system error handler, *OEThrow*. For example, this class may be implemented to customize how OpenEye toolkit messages are handled as all OpenEye toolkits will write their messages to *OEThrow*. The handler used by *OEThrow* can be changed with the *OEErrorHandler.SetHandlerImpl* method.

Implementations of this class only need to implement *OEErrorHandlerImplBase.Msg* and *OEErrorHandlerImplBase.CreateCopy*.

Constructors

```
OEErrorHandlerImplBase()
```

Default and copy constructors.

Clear

```
void Clear()
```

Count

```
unsigned int Count(unsigned int level = 0) const
```

CreateCopy

```
OEErrorHandlerImplBase* CreateCopy() const = 0
```

GetLevel

```
unsigned int GetLevel() const
```

GetStrict

```
bool GetStrict() const
```

IncrementCount

```
void IncrementCount(unsigned int level)
```

Msg

```
void Msg(unsigned int level, const std::string &)=0
```

SetLevel

```
void SetLevel(unsigned int level)
```

SetStrict

```
void SetStrict(bool strict)
```

OEExponentColorGradient

```
class OEExponentColorGradient : public OEColorGradientBase
```

The *OEExponentColorGradient* class interpolates between colors that are defined by using the OEColorStop class. The `OEExponentColorGradient.TransformValue` method defines the transformation that is performed before returning an interpolated color associated with a floating point value. The *Figure: Example of various exponent color gradients* illustrates the difference between *OEExponentColorGradient* objects constructed with the same color stops but with different *exponent* values. An *OEExponentColorGradient* object constructed with *1.0 exponent* value is equivalent to an *OELinearColorGradient* object constructed with the same color stops.

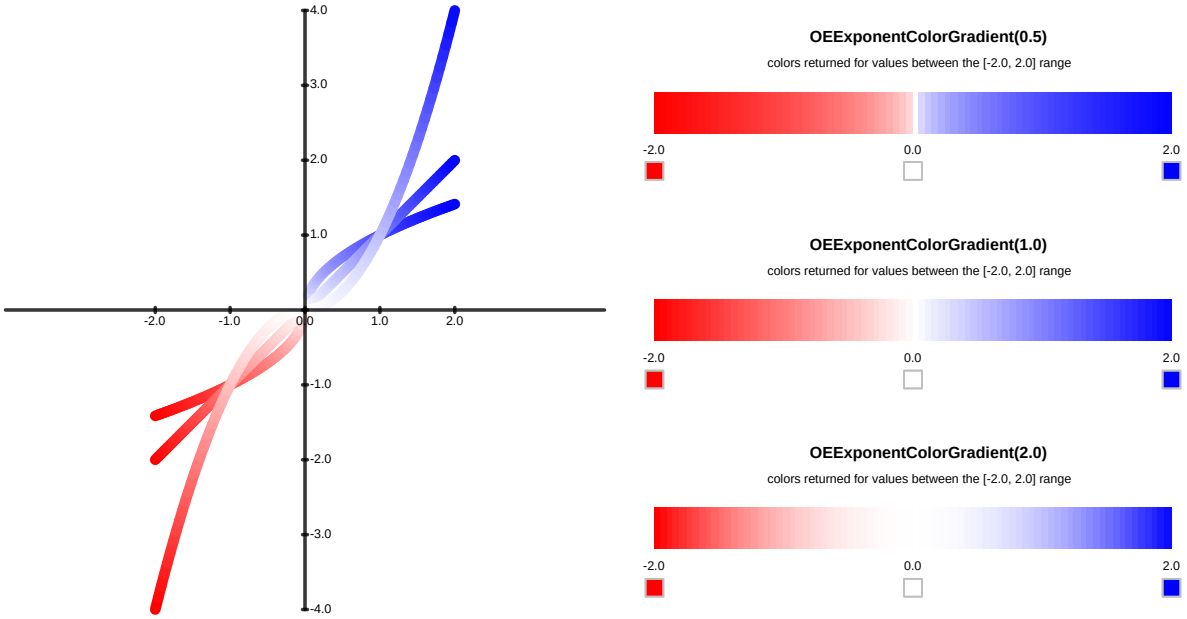


Fig. 23: **Example of various exponent color gradients** generated with (-2.0, OERed), (0.0, OEWhite) and (+2.0, OEBlue) color stops

See also

- *OEColorStop* class
- *OEExponentialColorGradient* class
- *OELinearColorGradient* class
- *OELogarithmicColorGradient* class

The following methods are publicly inherited from *OEColorGradientBase*:

operator=	*GetMaxColor*	*GetStops*
AddStop	*GetMaxValue*	*IsEmpty*
DeleteStop	*GetMinColor*	*NumStops*
GetColorAt	*GetMinValue*	*TransformValue*

Constructors

```
OEExponentColorGradient(double exponent)
```

Constructor that creates an "empty" color gradient with the given *exponent* value.

exponent
The given value is clamped into the range of `[0.1, 10.0]`.

> **Note**
>
> An "empty" color gradient always returns the *OEBlack* color, when calling the *OEColorGradientBase.GetColorAt*, the *OEColorGradientBase.GetMinColor* or the *OEColorGradientBase.GetMaxColor* methods.

```
OEExponentColorGradient(double exponent, const OEColorStop& oneStop)
```

Creates an *OEExponentColorGradient* object with one color stop and an *exponent* value. An *OEExponentColorGradient* object that has only one color stop will always return that color when calling the *OEColorGradientBase.GetColorAt* method.

exponent
The given value is clamped into the range of `[0.1, 10.0]`.

```
OEExponentColorGradient(double exponent,
                        const OEColorStop& bgnStop,
                        const OEColorStop& endStop)
```

Creates an *OEExponentColorGradient* object with interpolation range between the two specified color stops using the given *exponent* value.

exponent
The given value is clamped into the range of `[0.1, 10.0]`.

```
OEExponentColorGradient(const OEExponentColorGradient& rhs)
```

Copy constructors.

GetExponent

```
double GetExponent() const
```

Returns the *exponent* value used in the *OEExponentColorGradient.TransformValue* method.

SetExponent

```
void SetExponent(double exponent)
```

Sets the *exponent* value used in the *OEExponentColorGradient.TransformValue* method.

exponent
The given value is clamped into the range of `[0.1, 10.0]`.

TransformValue

```
double TransformValue(double value) const
```

Performs a $f(x) = -x^y$ transformation when x is less than 0.0 and $f(x) = x^y$ transformation when x is greater than 0.0, where the y is the exponent returned by *OEExponentColorGradient.GetExponent* method. This method is called by *OEColorGradientBase.GetColorAt* method to transform a value before performing color interpolation.

OEExponentialColorGradient

```
class OEExponentialColorGradient : public OEColorGradientBase
```

The *OEExponentialColorGradient* class interpolates between colors that are defined by using the OEColorStop class. The *OEExponentialColorGradient.TransformValue* method defines the transformation that is performed before returning an interpolated color associated with a floating point value. The *Figure: Example of linear and exponential color gradients* illustrates the difference between a linear and a logarithmic colors gradients constructed with the same color stops.

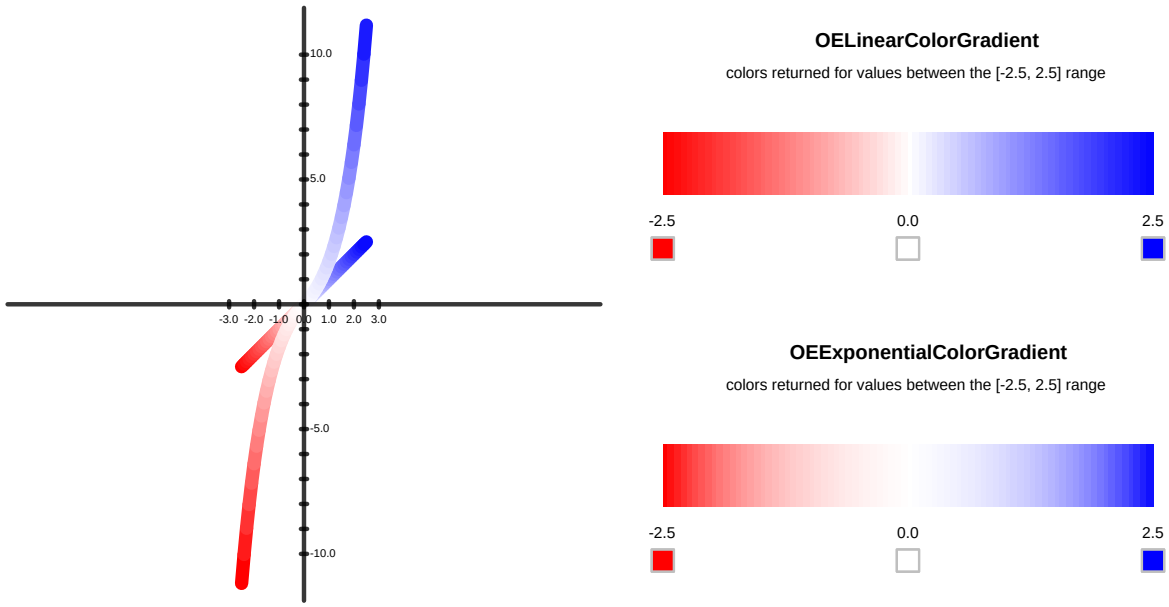


Fig. 24: **Example of linear and exponential color gradients** generated with (-2.5, OERed), (0.0, OEWhite) and (+2.5, OEBlue) color stops

See also

- *OEColorStop* class
- *OEExponentColorGradient* class
- *OELinearColorGradient* class
- *OELogarithmicColorGradient* class

The following methods are publicly inherited from *OEColorGradientBase*:

operator=	*GetMaxColor*	*GetStops*
AddStop	*GetMaxValue*	*IsEmpty*
DeleteStop	*GetMinColor*	*NumStops*
GetColorAt	*GetMinValue*	*TransformValue*

Constructors

```
OEExponentialColorGradient()
```

Default constructor that creates an "empty" color gradient.

> **Note**
>
> An "empty" color gradient always returns the *OEBlack* color, when calling the *OEColorGradientBase.GetColorAt*, the *OEColorGradientBase.GetMinColor* or the *OEColorGradientBase.GetMaxColor* methods.

```
OEExponentialColorGradient(const OEColorStop& oneStop)
```

Creates an *OEExponentialColorGradient* object with one color stop. An *OEExponentialColorGradient* object that has only one color stop will always return that color when calling the *OEColorGradientBase.GetColorAt* method.

```
OEExponentialColorGradient(const OEColorStop& bgnStop,
                           const OEColorStop& endStop)
```

Creates an *OEExponentialColorGradient* object with interpolation range between the two specified color stops.

```
OEExponentialColorGradient(const OEExponentialColorGradient& rhs)
```

Copy constructor.

TransformValue

```
double TransformValue(double value) const
```

Performs a $f(x) = -exp(-x) + 1.0$ transformation when x is less than 0.0 and $f(x) = exp(x) - 1.0)$ transformation when x is greater than 0.0. This method is called by *OEColorGradientBase.GetColorAt*

OEFixedGrid

```
template<class GridType>
class OEFixedGrid : public OEGridBase
```

This class represents *OEFixedGrid*.

The following methods are publicly inherited from *OEGridBase*:

Clear	*GetXMid*	*GetZMin*
CreateCopy	*GetXMin*	*GridIdxToElement*
ElementToGridIdx	*GetY*	*GridIdxToSpatialCoord*
ElementToSpatialCoord	*GetYDim*	*IsDataType*
GetDataType	*GetYIdx*	*IsInGrid*
GetDim	*GetYInc*	*IsSpacingSet*
GetMid	*GetYMax*	*IsTitleSet*
GetSize	*GetYMid*	*IsXMidSet*
GetSpacing	*GetYMin*	*IsYMidSet*
GetTitle	*GetZ*	*IsZMidSet*
GetX	*GetZDim*	*SetTitle*
GetXDim	*GetZIdx*	*SpatialCoordToElement*
GetXIdx	*GetZInc*	*SpatialCoordToGridIdx*
GetXInc	*GetZMax*	
GetXMax	*GetZMid*	

The following methods are publicly inherited from *OEGridAbstractBase*:

Clear	*GetTitle*	*GetZMid*
GetDataType	*GetXDim*	*IsDataType*
GetDim	*GetXMid*	*IsSpacingSet*
GetMid	*GetYDim*	*SetTitle*
GetSize	*GetYMid*	
GetSpacing	*GetZDim*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEGrid*
- *OEScalarGrid*

operator()

```
GridType& operator()(float x, float y, float z)
const GridType& operator()(float x, float y, float z) const
GridType& operator()(unsigned int ix, unsigned int iy, unsigned int iz)
const GridType& operator()(unsigned int ix, unsigned int iy, unsigned int iz) const
```

CreateCopy

```
OEBase* CreateCopy() const
```

GetDataType

```
const void *GetDataType() const
```

GetValues

```
GridType *GetValues()
const GridType *GetValues() const
```

IsDataType

```
bool IsDataType(const void *type) const
```

SetValues

```
bool SetValues(const GridType *dataptr, unsigned int length)
```

OEFixedMultiGrid

```
template<class GridType>
class OEFixedMultiGrid : public OEGridBase
```

This class represents *OEFixedMultiGrid*.

The following methods are publicly inherited from *OEGridBase*:

Clear	*GetXMid*	*GetZMin*
CreateCopy	*GetXMin*	*GridIdxToElement*
ElementToGridIdx	*GetY*	*GridIdxToSpatialCoord*
ElementToSpatialCoord	*GetYDim*	*IsDataType*
GetDataType	*GetYIdx*	*IsInGrid*
GetDim	*GetYInc*	*IsSpacingSet*
GetMid	*GetYMax*	*IsTitleSet*
GetSize	*GetYMid*	*IsXMidSet*
GetSpacing	*GetYMin*	*IsYMidSet*
GetTitle	*GetZ*	*IsZMidSet*
GetX	*GetZDim*	*SetTitle*
GetXDim	*GetZIdx*	*SpatialCoordToElement*
GetXIdx	*GetZInc*	*SpatialCoordToGridIdx*
GetXInc	*GetZMax*	
GetXMax	*GetZMid*	

The following methods are publicly inherited from *OEGridAbstractBase*:

Clear	*GetTitle*	*GetZMid*
GetDataType	*GetXDim*	*IsDataType*
GetDim	*GetXMid*	*IsSpacingSet*
GetMid	*GetYDim*	*SetTitle*
GetSize	*GetYMid*	
GetSpacing	*GetZDim*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEMultiGrid*

CreateCopy

```
OEBase* CreateCopy() const
```

GetDataType

```
const void *GetDataType() const
```

GetNumGrids

```
unsigned int GetNumGrids() const
```

GetValues

```
GridType *GetValues(unsigned int igrid = 0)
const GridType *GetValues(unsigned int igrid = 0) const
```

IsDataType

```
bool IsDataType(const void *type) const
```

SetNumGrids

```
bool SetNumGrids(unsigned int n)
```

OEGaussian

```
class OEGaussian : public OEGaussianBase
```

This class represents *OEGaussian*.

The following methods are publicly inherited from *OEGaussianBase*:

GetCenter	*GetY*	*SetWidth*
GetColor	*GetZ*	*SetX*
GetDataType	*IsDataType*	*SetY*
GetPrefactor	*SetCenter*	*SetZ*
GetWidth	*SetColor*	
GetX	*SetPrefactor*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEGaussian()
OEGaussian(const OEGaussian& rhs)
OEGaussian(const OEGaussianBase& rhs)
OEGaussian(float prefactor, float width, float* xyz = 0,
           unsigned int color = 0)
OEGaussian(float prefactor, float width, float x, float y, float z,
           unsigned int color = 0)
```

Default and copy constructors.

CreateCopy

```
OEBase* CreateCopy() const
```

GetCenter

```
void GetCenter(float* xyz) const
```

GetColor

```
unsigned int GetColor() const
```

GetDataType

```
const void* GetDataType() const
```

GetPrefactor

```
float GetPrefactor() const
```

GetWidth

```
float GetWidth() const
```

GetX

```
float GetX() const
```

GetY

```
float GetY() const
```

GetZ

```
float GetZ() const
```

IsDataType

```
bool IsDataType(const void *) const
```

SetCenter

```
void SetCenter(float* xyz)
```

SetColor

```
void SetColor(unsigned int color)
```

SetPrefactor

```
void SetPrefactor(float prefactor)
```

SetWidth

```
void SetWidth(float width)
```

SetX

```
void SetX(float x)
```

SetY

```
void SetY(float y)
```

SetZ

```
void SetZ(float z)
```

OEGaussianBase

```
class OEGaussianBase : public OEBase
```

This class represents *OEGaussianBase*.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEGaussian*

GetCenter

```
void GetCenter(float* xyz) const = 0
```

GetColor

```
unsigned int GetColor() const = 0
```

GetDataType

```
const void* GetDataType() const
```

GetPrefactor

```
float GetPrefactor() const = 0
```

GetWidth

```
float GetWidth() const = 0
```

GetX

```
float GetX() const = 0
```

GetY

```
float GetY() const = 0
```

GetZ

```
float GetZ() const = 0
```

IsDataType

```
bool IsDataType(const void *) const
```

SetCenter

```
void SetCenter(float* xyz)=0
```

SetColor

```
void SetColor(unsigned int color)=0
```

SetPrefactor

```
void SetPrefactor(float prefactor)=0
```

SetWidth

```
void SetWidth(float width)=0
```

SetX

```
void SetX(float x)=0
```

SetY

```
void SetY(float y)=0
```

SetZ

```
void SetZ(float z)=0
```

OEGaussianContainer

```
class OEGaussianContainer : public OEGaussianContainerBase
```

This class represents *OEGaussianContainer*.

The following methods are publicly inherited from *OEGaussianContainerBase*:

AddGaussian	*GetColors*	*GetPrefactors*
Clear	*GetCoords*	*GetWidths*
DeleteGaussian	*GetDataType*	*IsDataType*
DeleteGaussians	*GetGaussians*	*NumGaussians*

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEGaussianContainer()
OEGaussianContainer(const OEGaussianContainer &)
OEGaussianContainer(const OEGaussianContainerBase &)
```

Default and copy constructors.

AddGaussian

```
bool AddGaussian(const OEGaussianBase &)
```

Clear

```
void Clear()
```

CreateCopy

```
OEBase* CreateCopy() const
```

DeleteGaussian

```
bool DeleteGaussian(OEGaussianBase *)
```

DeleteGaussians

```
void DeleteGaussians()
```

GetColors

```
bool GetColors(unsigned int *) const
```

GetCoords

```
bool GetCoords(float *) const
bool GetCoords(double *) const
```

GetDataType

```
const void *GetDataType() const
```

GetGaussians

```
OESystem::OEIterBase<OEGaussianBase> *GetGaussians()
OESystem::OEIterBase<const OEGaussianBase> *GetGaussians() const
```

GetPrefactors

```
bool GetPrefactors(float *) const
bool GetPrefactors(double *) const
```

GetWidths

```
bool GetWidths(float *) const
bool GetWidths(double *) const
```

IsDataType

```
bool IsDataType(const void *) const
```

NumGaussians

```
unsigned int NumGaussians() const
```

OEGaussianContainerBase

```
class OEGaussianContainerBase  : public OEBase
```

This class represents *OEGaussianContainerBase*.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEGaussianContainer*

AddGaussian

```
bool AddGaussian(const OEGaussianBase &)=0
```

Clear

```
void Clear()
```

DeleteGaussian

```
bool DeleteGaussian(OEGaussianBase *)=0
```

DeleteGaussians

```
void DeleteGaussians()=0
```

GetColors

```
bool GetColors(unsigned int *) const = 0
```

GetCoords

```
bool GetCoords(float *) const = 0
bool GetCoords(double *) const = 0
```

GetDataType

```
const void* GetDataType() const
```

GetGaussians

```
OESystem::OEIterBase<OEGaussianBase>* GetGaussians()=0
OESystem::OEIterBase<const OEGaussianBase>* GetGaussians() const = 0
```

GetPrefactors

```
bool GetPrefactors(float *) const = 0
bool GetPrefactors(double *) const = 0
```

GetWidths

```
bool GetWidths(float *) const = 0
bool GetWidths(double *) const = 0
```

IsDataType

```
bool IsDataType(const void *) const
```

NumGaussians

```
unsigned int NumGaussians() const = 0
```

OEGrid

```
template<class GridType>
class OEGrid : public OEFixedGrid<GridType>
```

This class represents *OEGrid*.

The following methods are publicly inherited from *OEFixedGrid*:

operator()	*GetDataType*	*SetValues*
operator[]	*GetValues*	
CreateCopy	*IsDataType*	

The following methods are publicly inherited from *OEGridBase*:

Clear	*GetXMid*	*GetZMin*
CreateCopy	*GetXMin*	*GridIdxToElement*
ElementToGridIdx	*GetY*	*GridIdxToSpatialCoord*
ElementToSpatialCoord	*GetYDim*	*IsDataType*
GetDataType	*GetYIdx*	*IsInGrid*
GetDim	*GetYInc*	*IsSpacingSet*
GetMid	*GetYMax*	*IsTitleSet*
GetSize	*GetYMid*	*IsXMidSet*
GetSpacing	*GetYMin*	*IsYMidSet*
GetTitle	*GetZ*	*IsZMidSet*
GetX	*GetZDim*	*SetTitle*
GetXDim	*GetZIdx*	*SpatialCoordToElement*
GetXIdx	*GetZInc*	*SpatialCoordToGridIdx*
GetXInc	*GetZMax*	
GetXMax	*GetZMid*	

The following methods are publicly inherited from *OEGridAbstractBase*:

Clear	*GetTitle*	*GetZMid*
GetDataType	*GetXDim*	*IsDataType*
GetDim	*GetXMid*	*IsSpacingSet*
GetMid	*GetYDim*	*SetTitle*
GetSize	*GetYMid*	
GetSpacing	*GetZDim*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEScalarGrid*

Constructors

```
OEGrid()
```

Creates a one point *OEScalarGrid* grid object.

```
OEGrid(const OEGrid<GridType>& rhs)
OEGrid(const OEFixedGrid<GridType>& rhs)
```

Creates a *OEGrid* object and copies the entire contents of the 'rhs' grid.

```
OEGrid(const float minmax[6], float spacing, const char* title = "")
OEGrid(const double minmax[6], double spacing, const char* title = "")
```

Creates a *OEGrid* object from a minimum bounding box ('minmax'). The size of each length between grid points is determined by the 'spacing' parameter. The 'title' parameter is provided as a convenience for setting the title upon construction. It can later be retrieved with the *OEGridBase.GetTitle* method.

```
OEGrid(int xdim, int ydim, int zdim, float midx, float midy,
       float midz, float spacing, std::string title = "",
       GridType* dataptr = 0)
```

Creates and initializes a *OEGrid* object. The grid data can be copied from the optional parameter 'dataptr'. The pointer should be an array of size `xdim*ydim*zdim*sizeof(float)`.

Clear

```
void Clear()
```

Reallocates memory for the grid, effectively clearing all the grid values. Calls through to *OEBase.Clear*.

CreateCopy

```
OEBase* CreateCopy() const
```

GetDataType

```
const void *GetDataType() const
```

IsDataType

```
bool IsDataType(const void *type) const
```

SetAll

```
bool SetAll(unsigned int xdim, unsigned int ydim, unsigned int zdim,
            float midx, float midy, float midz, float spacing,
            std::string title, GridType *dataptr)
```

Convenience method for setting all the properties of this grid at once.

SetDim

```
bool SetDim(unsigned int ixdim, unsigned int iydim, unsigned int izdim)
```

Sets the number of grid points for each dimension.

SetMid

```
bool SetMid(float x, float y, float z)
```

Sets the mid-point (center) of the grid.

SetSpacing

```
bool SetSpacing(float spacing)
```

Sets the length between grid points.

SetXDim

```
bool SetXDim(unsigned int ixdim)
```

Sets the number of grid points in the x-dimension.

SetXMid

```
bool SetXMid(float x)
```

Sets the mid-point (center) in the x-dimension.

SetYDim

```
bool SetYDim(unsigned int iydim)
```

Sets the number of grid points in the y-dimension.

SetYMid

```
bool SetYMid(float y)
```

Sets the mid-point (center) in the y-dimension.

SetZDim

```
bool SetZDim(unsigned int izdim)
```

Sets the number of grid points in the z-dimension.

SetZMid

```
bool SetZMid(float z)
```

Sets the mid-point (center) in the z-dimension.

OEGridAbstractBase

```
class OEGridAbstractBase : public OEBase
```

This class represents *OEGridAbstractBase*.

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEFixedGrid*
- *OEFixedMultiGrid*
- *OEGrid*
- *OEGridBase*
- *OEMultiGrid*
- *OEScalarGrid*
- *OESkewGrid*

Clear

```
void Clear()
```

GetDataType

```
const void* GetDataType() const
```

GetDim

```
bool GetDim(unsigned int& xdim, unsigned int& ydim,
            unsigned int& zdim) const = 0
```

GetMid

```
bool GetMid(float& x, float& y, float& z) const = 0
```

GetSize

```
unsigned int GetSize() const = 0
```

GetSpacing

```
float GetSpacing() const = 0
```

GetTitle

```
std::string GetTitle() const = 0
```

GetXDim

```
unsigned int GetXDim() const = 0
```

GetXMid

```
float GetXMid() const = 0
```

GetYDim

```
unsigned int GetYDim() const = 0
```

GetYMid

```
float GetYMid() const = 0
```

GetZDim

```
unsigned int GetZDim() const = 0
```

GetZMid

```
float GetZMid() const = 0
```

IsDataType

```
bool IsDataType(const void* type) const
```

IsSpacingSet

```
bool IsSpacingSet() const = 0
```

SetTitle

```
bool SetTitle(const std::string& title)=0
```

OEGridBase

```
class OEGridBase : public OEGridAbstractBase
```

This class represents *OEGridBase*.

The following methods are publicly inherited from *OEGridAbstractBase*:

Clear	*GetTitle*	*GetZMid*
GetDataType	*GetXDim*	*IsDataType*
GetDim	*GetXMid*	*IsSpacingSet*
GetMid	*GetYDim*	*SetTitle*
GetSize	*GetYMid*	
GetSpacing	*GetZDim*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

The following classes derive from this class:

- *OEFixedGrid*
- *OEFixedMultiGrid*
- *OEGrid*
- *OEMultiGrid*
- *OEScalarGrid*

Clear

```
void Clear()
```

Reallocates memory for the grid, effectively clearing all the grid values. Calls through to *OEBase.Clear*.

CreateCopy

```
OEBase* CreateCopy() const
```

Makes a copy of the grid instance.

ElementToGridIdx

```
bool ElementToGridIdx(unsigned int el, unsigned int& ix,
                      unsigned int& iy, unsigned int& iz) const
```

The element 'el' is the one dimensional index into the grid points. This function converts this one dimensional index into the three dimensional indices (ix, iy, iz) for the associated grid point. Returns `false` if 'el' is outside the array bounds.

ElementToSpatialCoord

```
bool ElementToSpatialCoord(unsigned int el, float& x, float& y,
                           float& z) const
```

The element 'el' is the one dimensional index into the grid points. This function converts this one dimensional index into the Cartesian coordinates (x,y,z) of the associated grid point. Returns false if 'el' is outside the array bounds.

GetDataType

```
const void* GetDataType() const
```

GetDim

```
bool GetDim(unsigned int &xdim, unsigned int &ydim, unsigned int &zdim) const
```

Retrieves the grid dimensions (*i.e.* the wideness of the grid for a specific axis) into 'xdim', 'ydim', 'zdim' arguments.

GetMid

```
bool GetMid(float &x, float &y, float &z) const
```

Retrieves the grid midpoint (*i.e.* the center of the grid) into 'midx', 'midy' and 'midz' arguments. The midpoint defines where the grid is located in Cartesian space. Returns `false` if the midpoint is not completely defined (! *IsXMidSet()* || ! *IsYMidSet()* || ! *IsZMidSet()*)

GetSize

```
unsigned int GetSize() const
```

Returns the number of grid points inside this grid.

GetSpacing

```
float GetSpacing() const
```

Returns the length between grid points.

GetTitle

```
std::string GetTitle() const
```

Returns the title of this grid. The title defaults to an empty string.

GetX

```
float GetX(unsigned int ix) const
```

Returns the x-component of the Cartesian coordinate for the grid points with the x-index of 'ix'.

GetXDim

```
unsigned int GetXDim() const
```

Returns how many grid points there are for the x-dimension of this grid.

GetXIdx

```
unsigned int GetXIdx(float x) const
```

Returns the grid index in the x-dimension for the grid point identified by 'x', that corresponds to the x-component of the Cartesian coordinates.

```
unsigned int GetXIdx(unsigned int el) const
```

Returns the grid index in the x-dimension for the grid point identified by 'el', that corresponds to a specific point in the grid.

GetXInc

```
unsigned int GetXInc() const
```

Returns the increment to be added or subtracted to the grid point index to access the neighbor grid points along the x-axis.

GetXMax

```
float GetXMax() const
```

Returns the maximum extent of this grid along the x-axis.

GetXMid

```
float GetXMid() const
```

Returns the center of this grid along the x-axis.

GetXMin

```
float GetXMin() const
```

Returns the minimum extent of this grid along the x-axis.

GetY

```
float GetY(unsigned int iy) const
```

Returns the y-component of the Cartesian coordinate for the grid points with the y-index of 'iy'.

GetYDim

```
unsigned int GetYDim() const
```

Returns how many grid points there are for the y-dimension of this grid.

GetYIdx

```
unsigned int GetYIdx(float y) const
```

Returns the grid index in the y-dimension for the grid point identified by 'y', that corresponds to the y-component of the Cartesian coordinates.

```
unsigned int GetYIdx(unsigned int el) const
```

Returns the grid index in the y-dimension for the grid point identified by 'el', that corresponds to a specific point in the grid.

GetYInc

```
unsigned int GetYInc() const
```

Returns the increment to be added or subtracted to the grid point index to access the neighbor grid points along the y-axis.

GetYMax

```
float GetYMax() const
```

Returns the maximum extent of this grid along the y-axis.

GetYMid

```
float GetYMid() const
```

Returns the center of this grid along the y-axis.

GetYMin

```
float GetYMin() const
```

Returns the minimum extent of this grid along the y-axis.

GetZ

```
float GetZ(unsigned int iz) const
```

Returns the z-component of the Cartesian coordinate for the grid points with the z-index of ‘iz’.

GetZDim

```
unsigned int GetZDim() const
```

Returns how many grid points there are for the z-dimension of this grid.

GetZIdx

```
unsigned int GetZIdx(float z) const
```

Returns the grid index in the z-dimension for the grid point identified by ‘z’, that corresponds to the z-component of the Cartesian coordinates.

```
unsigned int GetZIdx(unsigned int el) const
```

Returns the grid index in the z-dimension for the grid point identified by ‘el’, that corresponds to a specific point in the grid.

GetZInc

```
unsigned int GetZInc() const
```

Returns the increment to be added or subtracted to the grid point index to access the neighbor grid points along the z-axis.

GetZMax

```
float GetZMax() const
```

Returns the maximum extent of this grid along the z-axis.

GetZMid

```
float GetZMid() const
```

Returns the center of this grid along the z-axis.

GetZMin

```
float GetZMin() const
```

Returns the minimum extent of this grid along the z-axis.

GridIdxToElement

```
bool GridIdxToElement(unsigned int ix, unsigned int iy,
                      unsigned int iz, unsigned int& el) const
```

Converts the three dimensional grid indices (ix,iy,iz) into the one dimensional grid point index 'el'. Returns `false` if (ix,iy,iz) are outside the grid bounds.

See also

- *OEGridBase.ElementToGridIdx*

GridIdxToSpatialCoord

```
bool GridIdxToSpatialCoord(unsigned int ix, unsigned int iy,
                           unsigned int iz, float& x, float& y,
                           float& z) const
```

Converts the three dimensional grid indices (ix,iy,iz) into the Cartesian coordinates (x,y,z) of the associated grid point. Returns false if (ix,iy,iz) are outside the grid bounds.

See also

- *OEGridBase.SpatialCoordToGridIdx*

IsDataType

```
bool IsDataType(const void* type) const
```

Returns whether type is the same as the instance this method is called on.

IsInGrid

```
bool IsInGrid(float x, float y, float z) const
bool IsInGrid(unsigned int ix, unsigned int iy, unsigned int iz) const
```

Returns whether the given coordinates are located within the grid bounds.

IsSpacingSet

```
bool IsSpacingSet() const
```

Returns whether the spacing for this grid has been specified.

IsTitleSet

```
bool IsTitleSet() const
```

Returns whether there is a title associated with this grid.

IsXMidSet

```
bool IsXMidSet() const
```

Returns whether the midpoint for this grid has been specified for the x-dimension.

IsYMidSet

```
bool IsYMidSet() const
```

Returns whether the midpoint for this grid has been specified for the y-dimension.

IsZMidSet

```
bool IsZMidSet() const
```

Returns whether the midpoint for this grid has been specified for the z-dimension.

SetTitle

```
bool SetTitle(const std::string& title)
```

Sets the title associated with this grid.

SpatialCoordToElement

```
bool SpatialCoordToElement(float x, float y, float z,
                           unsigned int& el) const
```

Converts the Cartesian coordinates (x,y,z) into the grid point index 'el'. Returns `false` if (x,y,z) are outside the grid bounds.

See also

- *OEGridBase.ElementToSpatialCoord*

SpatialCoordToGridIdx

```
bool SpatialCoordToGridIdx(float x, float y, float z,
                           unsigned int& ix, unsigned int& iy,
                           unsigned int& iz) const
```

Converts the Cartesian coordinates (x,y,z) into the three dimensional indices (ix,iy,iz). Returns `false` if (x,y,z) are outside the grid bounds.

See also

- *OEGridBase.GridIdxToSpatialCoord*

OEHasRole

```
class OEHasRole : public OESystem::OEUnaryPredicate<OERoleSet>
```

This class represents *OEHasRole*, a functor for objects inherited from *OERoleSet* that identifies those containing a specified role (*OERole*).

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasRole(const OERole& role)
```

Copy constructor.

```
OEHasRole(const std::string& name)
```

Convenience constructor allowing the predicate to be constructed from a *role name* string.

operator()

```
bool operator()(const OERoleSet& myObject) const
```

Returns `true` if the *OERoleSet* contains an *OERole* that matches the value used in the *OEHasRole.Constructors*.

CreateCopy

```
OESystem::OEUnaryFunction<OERoleSet , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasRole* object is dynamically allocated and owned by the caller.

OEHeader

```
class OEHeader : public OEContainer
```

This class represents *OEHeader*.

The following methods are publicly inherited from *OEContainer*:

operator=	*GetDataType*
CreateCopy	*IsDataType*

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEHeader()
OEHeader(int argc, const char* argv[], const char* appname,
         const char* version, const char* comment = "",
         const OEInterface& itf = OEInterface(),
         const OEHeader* history = 0x0)
OEHeader(const OEHeader& rhs)
```

Default and copy constructors.

AddHistory

```
void AddHistory(const OEHeader& header)
```

GetCommandLine

```
std::string GetCommandLine() const
```

GetComment

```
std::string GetComment() const
```

GetDate

```
void GetDate(int& day, int& month, int& year) const
void GetDate(int& day, int& month, int& year, int& hour,
             int& minute, int& seconds) const
```

GetHistory

```
OEIterBase<OEHeader>* GetHistory() const
```

GetInterface

```
std::string GetInterface() const
```

GetOERelease

```
std::string GetOERelease() const
```

GetOEVersion

```
int GetOEVersion() const
```

GetPlatform

```
std::string GetPlatform() const
```

GetProgram

```
std::string GetProgram() const
```

GetVersion

```
std::string GetVersion() const
```

SetCommandLine

```
void SetCommandLine(const std::string& comment)
```

SetComment

```
void SetComment(const std::string& comment)
```

SetDate

```
void SetDate()
void SetDate(int day, int month, int year)
void SetDate(int day, int month, int year, int hour, int minute,
             int seconds)
```

SetInterface

```
void SetInterface(const OEInterface& itf)
```

SetOERelease

```
void SetOERelease(const std::string& release)
```

SetOEVersion

```
void SetOEVersion(int version)
```

SetPlatform

```
void SetPlatform(const std::string& platform)
```

SetProgram

```
void SetProgram(const std::string& program)
```

SetVersion

```
void SetVersion(const std::string &)
```

OEInterface

```
class OEInterface
```

This class represents *OEInterface*.

Constructors

```
OEInterface()
OEInterface(const OEInterface &)
```

Default and copy constructors.

```
OEInterface(const unsigned char* dataptr)
OEInterface(const char* dataptr)
```

Constructs an *OEInterface* and configures it with the interface definition passed by `dataptr`. This is equivalent to the following code:

> **Note**
>
> It is possible to configure an *OEInterface* from multiple interface definitions. Just call *OEConfigure* once per interface definition on the same *OEInterface* object.

```
OEInterface(const unsigned char* dataptr, int argc, char** argv)
OEInterface(const char* dataptr, int argc, char** argv)
```

Constructs an *OEInterface* and configures it with the interface definition passed by `dataptr`. Then parses the command line given by `argv`. This is equivalent to the following code:

> **Warning**
>
> These convenience constructors may force the program to exit and thus may not be appropriate when writing a library that should instead return errors.

AddDetail

```
bool AddDetail(const std::string& detail)
```

Adds a line to the detailed description of the *OEInterface*. Returns `true` if the add was successful and `false` otherwise. (The current implementation never fails)

AddInterface

```
OEInterface* AddInterface()
```

Adds a child *OEInterface* and returns a pointer to the newly created object (which is owned by the current *OEInterface* object).

AddParameter

```
OEParameter* AddParameter(std::string classid)
```

Adds a parameter to the *OEInterface* instance. The returned pointer points to the newly created *OEParameter* object. This memory is **owned** by the *OEInterface* object. If the add fails `NULL` will be returned.

The implementation of the returned *OEParameter* pointer will always be a subclass of

```
template <TT> OETypedParameter
```

The specific type depends upon the value of classid as follows :

string
: TT is `std::string` parameter

double
: TT is `double` parameter

float
: TT is `float` parameter

bool
: TT is `bool` parameter

int
: TT is `int` parameter

file
: TT is *oeisstream* parameter

param_file
: TT is *oeisstream* and additionally the parameter is specially recognized by *OEParseCommandLine* as a text file holding parameter settings.

Clear

```
bool Clear()
```

Returns the *OEInterface* object to the default constructed state of an *OEInterface*.

DeleteInterface

```
bool DeleteInterface(OEInterface& itf)
```

Deletes a child *OEInterface* object. 'itf' is the child *OEInterface* to be deleted. The method returns `true` if the deletion is successful and `false` otherwise. 'itf' must be directly owned by the *OEInterface*, not one of its child *OEInterfaces*.

DeleteParameter

```
bool DeleteParameter(OEParameter& param)
```

Deletes a parameter ('param') owned by this *OEInterface* object. The function returns `true` if the delete succeeds and `false` otherwise. The parameter must be **owned** by the *OEInterface* object to be deleted (it will not delete parameters that belong to child *OEInterface* objects of this *OEInterface*).

Factory

```
static OESimpleFactory<std::string, OEParameter>& Factory()
```

Returns the *OEInterface* class's factory used to create new parameters.

Get

```
template <class TT>
TT Get(std::string name, unsigned int ilist = 0) const
```

Returns the setting (either the value, or default if the value is not set) of the parameter with the given name and instance. The parameter must be of type `TT`, and must have either have a value or default set. If the parameter cannot be found (*i.e.*, a call to *OEInterface.Has* with the same `TT` and 'name' returns `false`) a warning is issued and a default constructed `TT` is returned.

GetBrief

```
std::string GetBrief() const
```

Returns the brief description of the *OEInterface* object.

GetDetail

```
OEIterBase<const std::string>* GetDetail() const
```

Returns the detailed description of the *OEInterface* object.

GetInterface

```
OEInterface* GetInterface(std::string name)
const OEInterface* GetInterface(std::string name) const
```

Returns a pointer to a child *OEInterface* with a given 'name' owned by any child *OEInterface*. The search is recursive searching through all child's child *OEInterfaces*, etc. If no such *OEInterface* exists NULL is returned.

GetInterfaces

```
OEIterBase< OEInterface>* GetInterfaces(bool recursive = false)
OEIterBase<const OEInterface>* GetInterfaces(bool recursive = false) const
```

Returns and iterator over all child *OEInterfaces* of this object. If recursive is true, the iterator will include all of the children's *OEInterfaces* recursively in arbitrary order. If recursive is false only the immediate child *OEInterfaces* will be returned, and they will be sorted by *OEInterface.GetOrderPriority* (lowest values first) and then name.

GetList

```
template <class TT>
OEIterBase<const TT>* GetList(std::string name) const
```

See also

- Example program *CatMols.cs*

GetName

```
std::string GetName() const
```

Returns the name of the *OEInterface* object.

GetOrderPriority

```
int GetOrderPriority() const
```

Returns the order priority of the *OEInterface* object.

GetParameter

```
OEParameter* GetParameter(std::string name)
const OEParameter* GetParameter(std::string name) const
```

Returns a pointer to a *OEParameter* with a given 'name' owned by the *OEInterface* object or one of its child *OEInterfaces* (or child's child *OEInterface*, etc). If no such parameter exists NULL is returned.

GetParameters

```
OEIterBase<OEParameter>* GetParameters(bool recursive = true)
OEIterBase<const OEParameter>* GetParameters(bool recursive = true) const
```

Returns an iterator over all *OEParameters* of the *OEInterface* object. If recursive is `true` the search is recursive including parameters held by all child *OEInterfaces* and their children in arbitrary order. If recursive is `false` parameters held by child *OEInterfaces* will not be included and they will be sorted by the parameters order priority (lowest values first), followed by name.

GetTypedParameter

```
template<class TT>
OETypedParameter<TT> *GetTypedParameter(std::string name)
template<class TT>
const OETypedParameter<TT> *GetTypedParameter(std::string name) const
```

Has

```
template<class TT>
bool Has(std::string name, unsigned int ilist = 0) const
```

Returns `true` if all of the following are `true`:

1. A parameter with name *name* is contained by the *OEInterface*.
2. The parameter is of type `TT`
3. The parameters value of default is set.

In all other cases the function returns `false`.

Parent

```
OEInterface* Parent()
const OEInterface* Parent() const
```

Returns a pointer to the parent *OEInterface* of this object, or `NULL` if this *OEInterface* object does not have a parent.

SetBrief

```
bool SetBrief(std::string brief)
```

Sets the brief description of the *OEInterface*. Returns `true` if the brief was successfully set and `false` otherwise. (The current implementation never fails)

SetName

```
bool SetName(std::string name)
```

Sets the name of the *OEInterface*. Returns `true` if the name was successfully set and `false` otherwise. (The current implementation never fails).

SetOrderPriority

```
bool SetOrderPriority(int priority)
```

Sets the order priority of the *OEInterface*. The order priority overrides the default sorting by name that is used when calling the *GetInterfaces(false)* member function. Child *OEInterfaces* with lower priority values will appear first. This function returns `true` if the priority was successfully set and `false` otherwise. (The current implementation never fails).

OEIsFalse

```
template<class Arg> struct OEIsFalse : public OEUnaryPredicate<Arg>
```

This functor always returns `false` for any argument passed to it. The following specializations are provided to make using this class easier:

- `OEIsFalseAtom`
- `OEIsFalseBond`
- `OEIsFalseConf`

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following classes derive from this class:

- *IsFalse*
- *OEUnaryFalse*

operator()

```
bool operator()(const Arg &) const
```

Always returns `false`.

CreateCopy

```
OEUnaryFunction<Arg, bool>* CreateCopy() const
```

OEIsTrue

```
template<class Arg> struct OEIsTrue : public OEUnaryPredicate<Arg>
```

This functor always returns `true` for any argument passed to it. The following specializations are provided to make using this class easier:

- `OEIsTrueAtom`

- OEIsTrueBond
- OEIsTrueConf

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following classes derive from this class:

- *IsTrue*
- *OEUnaryTrue*

operator()

```
bool operator()(const Arg &) const
```

Always returns `true`.

CreateCopy

```
OEUnaryFunction<Arg, bool>* CreateCopy() const
```

OEIter

```
template<class A>
class OEIter
```

This class represents *OEIter*.

The *OEIter* template class is used to represent *iterator* functionality. The template is parameterized over `class A`, defining the behavior of iterators of across collections of type `A`. The iterator can be thought of a behaving like a pointer to type `A`.

OEIters are typically initialized by assigning them the value of an *OEIterBase(of compatible type T)*, return from a function.

Constructors

```
OEIter()
OEIter(OEIterBase<A>* rhs)
OEIter(const OEIter<A>& rhs)
OEIter(OEIterBase<NonConstA>* rhs)
template<typename P>
OEIter(const P& pred, OEIter<A>& rhs)
template<typename P>
OEIter(const P& pred, OEIterBase<A>* rhs)
```

Default and copy constructors.

operator->

```
A* operator->() const
```

Dereferences the iterator as a pointer. This operator returns a pointer to the current item pointed to by the iterator.

operator bool

```
operator bool() const
```

Returns true if the current iterator position is valid. This should typically be checked before each dereference of an iterator.

Copy

```
OEIterBase<A> *Copy() const
```

Copies an *OEIter*. The returned value is a newly allocated *OEIterBase* <A>* that should be immediately assigned to an *OEIter*. The current location in the iterator is preserved by the copy.

Push

```
void Push(OEIter<A> &iter)
void Push(OEIterBase<A> *ib)
template<typename F>
void Push(const F &pred)
```

Sort

```
void Sort(const OESystem::OEBinaryPredicate<A, A> &sort_criterion)
```

Sorts contents of the iterator and resets the iterator to its first item. The re-ordering is local to the given instance of the iterator (*i.e.*, the container holding the items the iterator traverses is unchanged). Once this function is called the iterator is never delete safe, even if it normally would be.

The *OEBinaryPredicate* <A,A> passed to this function is used as the less comparison in the sort. It should return `true` if the first argument is less than the second argument and `false` otherwise. An item that is less than another item will appear first after sorting.

ToFirst

```
OEIter<A> &ToFirst()
```

Resets the iterator to its first item, if possible.

ToLast

```
OEIter<A> &ToLast()
```

Resets the iterator to its last item, if possible.

OEIterBase

```
template<class A>
class OEIterBase
```

This class represents *OEIterBase*.

The *OEIterBase* class is an internal abstract template class. All iterators returned by *OESystem* and *OEChem* are pointers to instances derived from this class. However, users should not use or access this class/template directly. Instead the *OEIterBase* pointer should be assigned to an *OEIter*. The *OEIter*, for example, ensures that the contents of the *OEIterBase* will be destroyed when the *OEIter* goes out of scope, thus preventing potential memory leaks.

The following classes derive from this class:

- *OEConcatIter*
- *OEPredIter*

Constructors

```
OEIterBase()
```

Default and copy constructors.

operator->

```
A* operator->() const = 0
```

operator bool

```
operator bool() const
```

Copy

```
OEIterBase<A> *Copy() const = 0
```

Push

```
OEIterBase<A> *Push(OEIterBase<A> *ib)
```

ToFirst

```
OEIterBase<A> &ToFirst()=0
```

ToLast

```
OEIterBase<A> &ToLast()=0
```

OELinearColorGradient

```
class OELinearColorGradient : public OEColorGradientBase
```

The *OELinearColorGradient* class interpolates between colors that are defined by using the OEColorStop class. See *Figure Example of linear color gradient*

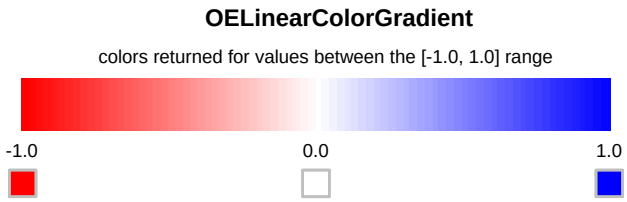


Fig. 25: **Example of linear color gradient** with (-1.0, OERed), (0.0, OEWhite) and (+1.0, OEBlue) color stops

See also

- *OEColorStop* class
- *OEExponentColorGradient* class
- *OEExponentialColorGradient* class
- *OELogarithmicColorGradient* class

The following methods are publicly inherited from *OEColorGradientBase*:

operator=	*GetMaxColor*	*GetStops*
AddStop	*GetMaxValue*	*IsEmpty*
DeleteStop	*GetMinColor*	*NumStops*
GetColorAt	*GetMinValue*	*TransformValue*

Constructors

```
OELinearColorGradient()
```

Default constructor that creates an “empty” color gradient.

Note

An “empty” color gradient always returns the *OEBlack* color, when calling the *OEColorGradientBase.GetColorAt*, the *OEColorGradientBase.GetMinColor* or the *OEColorGradientBase.GetMaxColor* methods.

```
OELinearColorGradient(const OEColorStop& oneStop)
```

Creates an *OELinearColorGradient* object with one color stop. An *OELinearColorGradient* object that has only one color stop will always return that color when calling the *OEColorGradientBase.GetColorAt* method.

```
OELinearColorGradient(const OEColorStop& bgnStop,
                      const OEColorStop& endStop)
```

Creates an *OELinearColorGradient* object with interpolation range between the two specified color stops.

```
OELinearColorGradient(const OELinearColorGradient& rhs)
```

Copy constructor.

TransformValue

```
double TransformValue(double value) const
```

Performs a $f(x) = x$ linear transformation. This method is called by *OEColorGradientBase.GetColorAt* method to transform a value before performing color interpolation.

OELogarithmicColorGradient

```
class OELogarithmicColorGradient : public OEColorGradientBase
```

The *OELogarithmicColorGradient* class interpolates between colors that are defined by using the OEColorStop class. The *OELogarithmicColorGradient.TransformValue* method defines the transformation that is performed before returning an interpolated color associated with a floating point value. The *Figure: Example of linear and logarithmic color gradients* illustrates the difference between a linear and a logarithmic colors gradients constructed with the same color stops.

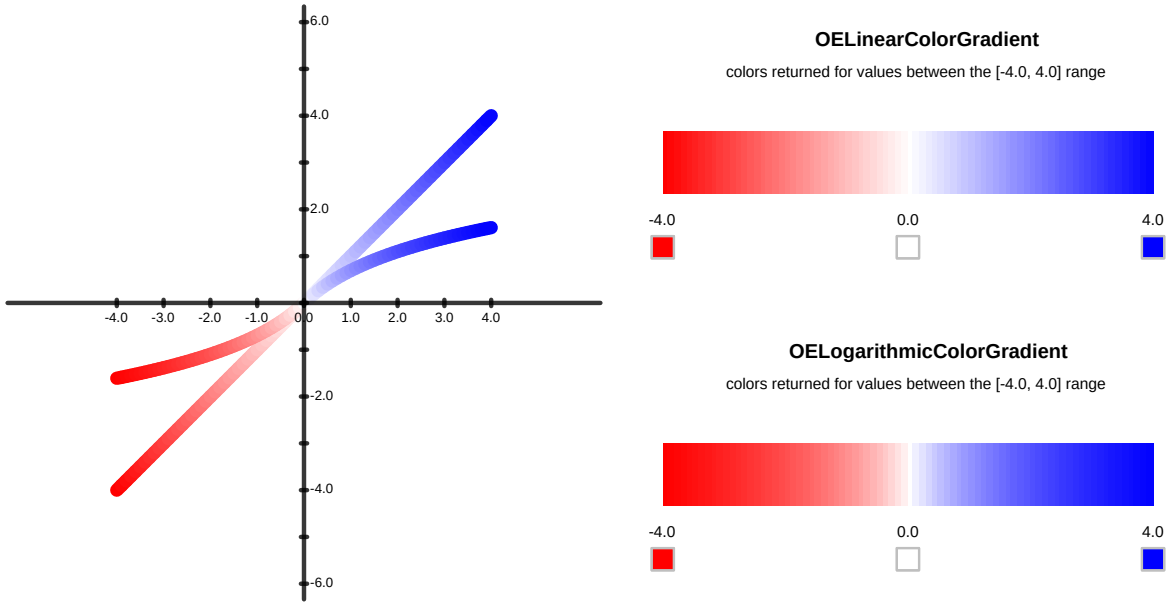


Fig. 26: **Example of linear and logarithmic color gradients** generated with (-4.0, OERed), (0.0, OEWhite) and (+4.0, OEBlue) color stops

See also

- *OEColorStop* class
- *OEExponentColorGradient* class
- *OEExponentialColorGradient* class
- *OELinearColorGradient* class

This class represents *OELogarithmicColorGradient*.

The following methods are publicly inherited from *OEColorGradientBase*:

operator=	*GetMaxColor*	*GetStops*
AddStop	*GetMaxValue*	*IsEmpty*
DeleteStop	*GetMinColor*	*NumStops*
GetColorAt	*GetMinValue*	*TransformValue*

Constructors

```
OELogarithmicColorGradient()
```

Default constructor that creates an "empty" color gradient.

Note

An "empty" color gradient always returns the *OEBlack* color, when calling the *OEColorGradientBase.GetColorAt*, the *OEColorGradientBase.GetMinColor* or the *OEColorGradientBase.GetMaxColor* methods.

```
OELogarithmicColorGradient(const OEColorStop& oneStop)
```

Creates an *OELogarithmicColorGradient* object with one color stop. An *OELogarithmicColorGradient* object that has only one color stop will always return that color when calling the *OEColorGradientBase.GetColorAt* method.

```
OELogarithmicColorGradient(const OEColorStop& bgnStop,
                           const OEColorStop& endStop)
```

Creates an *OELogarithmicColorGradient* object with interpolation range between the two specified color stops.

```
OELogarithmicColorGradient(const OELogarithmicColorGradient& rhs)
```

Copy constructor.

TransformValue

```
double TransformValue(double value) const
```

Performs a $f(x) = -log(-x + 1.0)$ transformation when x is less than 0.0 and $f(x) = log(x + 1.0)$ transformation when x is greater than 0.0. This method is called by *OEColorGradientBase.GetColorAt* method to transform a value before performing color interpolation.

OEMultiGrid

```
template<class GridType>
class OEMultiGrid : public OEFixedMultiGrid<GridType>
```

This class represents *OEMultiGrid*.

The following methods are publicly inherited from *OEFixedMultiGrid*:

operator[]	*GetNumGrids*	*SetNumGrids*
CreateCopy	*GetValues*	
GetDataType	*IsDataType*	

The following methods are publicly inherited from *OEGridBase*:

Clear	*GetXMid*	*GetZMin*
CreateCopy	*GetXMin*	*GridIdxToElement*
ElementToGridIdx	*GetY*	*GridIdxToSpatialCoord*
ElementToSpatialCoord	*GetYDim*	*IsDataType*
GetDataType	*GetYIdx*	*IsInGrid*
GetDim	*GetYInc*	*IsSpacingSet*
GetMid	*GetYMax*	*IsTitleSet*
GetSize	*GetYMid*	*IsXMidSet*
GetSpacing	*GetYMin*	*IsYMidSet*
GetTitle	*GetZ*	*IsZMidSet*
GetX	*GetZDim*	*SetTitle*
GetXDim	*GetZIdx*	*SpatialCoordToElement*
GetXIdx	*GetZInc*	*SpatialCoordToGridIdx*
GetXInc	*GetZMax*	
GetXMax	*GetZMid*	

The following methods are publicly inherited from *OEGridAbstractBase*:

Clear	*GetTitle*	*GetZMid*
GetDataType	*GetXDim*	*IsDataType*
GetDim	*GetXMid*	*IsSpacingSet*
GetMid	*GetYDim*	*SetTitle*
GetSize	*GetYMid*	
GetSpacing	*GetZDim*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEMultiGrid()
OEMultiGrid(const OEMultiGrid<GridType>& rhs)
OEMultiGrid(const OEFixedMultiGrid<GridType>& rhs)
OEMultiGrid(const float minmax[6], float spacing, int ngrids,
            const char* title = "")
OEMultiGrid(const double minmax[6], double spacing, int ngrids,
            const char* title = "")
OEMultiGrid(int xdim, int ydim, int zdim, float midx, float midy,
            float midz, float spacing, int ngrids, std::string title,
            GridType* dataptr)
```

Default and copy constructors.

Clear

```
void Clear()
```

CreateCopy

```
OEBase* CreateCopy() const
```

GetDataType

```
const void *GetDataType() const
```

IsDataType

```
bool IsDataType(const void *type) const
```

SetDim

```
bool SetDim(unsigned int ixdim, unsigned int iydim, unsigned int izdim)
```

SetMid

```
bool SetMid(float x, float y, float z)
```

SetSpacing

```
bool SetSpacing(float spacing)
```

SetValues

```
bool SetValues(unsigned int igrid, const GridType *dataptr,
               unsigned int length)
```

SetXDim

```
bool SetXDim(unsigned int ixdim)
```

SetXMid

```
bool SetXMid(float x)
```

SetYDim

```
bool SetYDim(unsigned int iydim)
```

SetYMid

```
bool SetYMid(float y)
```

SetZDim

```
bool SetZDim(unsigned int izdim)
```

SetZMid

```
bool SetZMid(float z)
```

OENot

```
template<class Arg>
class OENot : public OEUnaryPredicate<Arg>
```

This class represents *OENot*.

This class template is a composition predicate which generates a predicate which is the logical *not* operation of the predicate passed as argument.

See also

- *Composite Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following specializations exist for this template :

OENotAtom	OESystem::OENot<OEChem::OEAtomBase>
OENotBond	OESystem::OENot<OEChem::OEBondBase>
OENotGroup	OESystem::OENot<OEChem::OEGroupBase>
OENotConf	OESystem::OENot<OEChem::OEConfBase>
OENotRoleSet	OESystem::OENot<OEChem::OERoleSet>
OENotInteractionHint	OESystem::OENot<OEChem::OEInteractionHint>

Constructors

```
OENot()
OENot(const OENot<Arg>& rhs)
OENot(const OEUnaryPredicate<Arg>& func)
```

Default and copy constructors.

operator()

```
bool operator()(const Arg& arg) const
```

CreateCopy

```
OEUnaryFunction<Arg, bool>* CreateCopy() const
```

OEOr

```
template<class Arg>
class OEOr : public OEUnaryPredicate<Arg>
```

This class represents *OEOr*.

This class template is a composition predicate which generates a predicate which is the logical *or* operation of the two predicates passed as arguments.

See also

- *Composite Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

The following specializations exist for this template :

OEOrAtom	OESystem::OEOr<OEChem::OEAtomBase>
OEOrBond	OESystem::OEOr<OEChem::OEBondBase>
OEOrConf	OESystem::OEOr<OEChem::OEConfBase>
OEOrGroup	OESystem::OEOr<OEChem::OEGroupBase>
OEOrRoleSet	OESystem::OEOr<OEChem::OERoleSet>
OEOrInteractionHint	OESystem::OEOr<OEChem::OEInteractionHint>

Constructors

```
OEOr()
OEOr(const OEOr<Arg>& rhs)
OEOr(const OEUnaryPredicate<Arg>& func1,
     const OEUnaryPredicate<Arg>& func2)
```

Default and copy constructors.

operator()

```
bool operator()(const Arg& arg) const
```

CreateCopy

```
OEUnaryFunction<Arg, bool>* CreateCopy() const
```

OEOwnedPtr

```
template<class T>
class OEOwnedPtr
```

Warning

This class has been deprecated. Use `std::unique_ptr` instead. C++ only class. This class is not visible in any other languages.

This class is a classic smart pointer used for automatically managing the dynamic memory allocated for the object `T`. Explicitly calling `delete` is generally frowned upon in modern C++ due to the propensity of explicit `delete` leading to resource leaks and crashes. Deleting memory should be handled more automatically by mechanisms like *OEOwnedPtr* and *OESharedPtr*.

OEOwnedPtr is designed to mimic the C++11 std::unique_ptr in functionality with some important differences.

The first and most obvious difference is that *OEOwnedPtr* will compile and work in non-C++11 compilers. This provides the convenience and safety of std::unique_ptr without having to wait for every *OpenEye* platform to support C++11. That means the following will compile on all *OpenEye* supported platforms:

```
struct Foo { };

struct BarOwnsFoo
{
```

(continues on next page)

(continued from previous page)

```
  OEOwnedPtr<Foo> _owned;
  Foo *GetFoo() { return _owned; }
};
```

Note

Users are free to use C++11 features with the OpenEye toolkits in their own code. OpenEye will be slowly phasing in more support for C++11 features over many releases. For example, *OEOwnedPtr* has move constructors to allow for the following idiom to work in C++11 compilers:

```
OEOwnedPtr<Foo> MakeFoo()
{
  return OEOwnedPtr<Foo>(new Foo);
};
```

The second key difference between *OEOwnedPtr* and std::unique_ptr is whether `const` is maintained when accessing the object being pointed to. By default, in order to maintain compatibility with C, C++ allows non-const access to const pointers. By extension, this same rule applied to C++ smart pointers. This leads to very surprising behavior like the following:

```
struct Foo
{
  int _x;
  void AlterFoo() { _x = 1; }
};

struct BarOwnsFoo
{
  Foo *_foo;
  void AConstMethod() const { _foo->AlterFoo(); }
};
```

Even though `AConstMethod` is `const`, it isn't actually logically const. The internals of the class are changing. This behavior may seem benign at first, but it has vast implications in a library seeking to be thread-safe. The `const` on a method implies that the method is thread-safe, especially in C++11. In order to enforce thread-safety at runtime, *OEOwnedPtr* maintains the "const'ness" of its accessor methods. Therefore, the following is actually a compile error:

```
struct Foo
{
  int _x;
  void AlterFoo() { _x = 1; }
};

struct BarOwnsFoo
{
  OEOwnedPtr<Foo> _foo;
  void AConstMethod() const { _foo->AlterFoo(); }
};
```

Catching thread-safety problems very early, at compilation, is highly desirable for *OpenEye*. Hence the reason to differ from the standard smart pointer semantics in this way.

Constructors

```
OEOwnedPtr()
OEOwnedPtr(T* ptr)
```

Default constructors the pointer to NULL. Or constructs a smart pointer around the pointer ptr.

```
OEOwnedPtr(OEOwnedPtr&& rhs)
```

Move constructor to safely transfer ownership of the underlying pointer from one *OEOwnedPtr* to another. This allows *OEOwnedPtr* to be usable in container objects like std::vector in C++11 compatible compilers.

Get

```
T* Get()
const T* Get() const
```

Access the pointer held by this smart pointer while maintaining const correctness.

Release

```
T* Release()
```

Returns the pointer owned by this smart pointer. The smart pointer will cease to "own" the pointer in question. The smart pointer will effectively become a NULL pointer.

> **Warning**
>
> The user is expected to delete the pointer, usually by passing to a different smarter pointer class immediately.

OEParameter

```
class OEParameter
```

The *OEParameter* is an abstract base class. The parameters represent indivudual options in a *OEOptions* class, that can eventually translate to a parameter in a command-line interface application.

The *OEParameter* class defines the following public methods:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *GetLegalValues* and *GetPrettyLegalValues*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*

- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

The following classes derive from this class:

- *OETypedParameter*
- *OEPointerParameter*

AddAlias

```
bool AddAlias(const std::string& alias)
```

Adds an 'alias' name for this parameter. The alias must be a single word and begin with a `'-'`. This function returns `true` if the alias is valid and the add succeeds.

AddDetail

```
bool AddDetail(const std::string& detail)
```

Adds a line to the detailed description of the parameter. Return `true` if completed successfully.

AddIllegalRange

```
bool AddIllegalRange(const std::string& hi_val,
                     const std::string& low_val)
```

Adds an illegal range for this parameter in string format. This function succeeds and returns `true` if the parameter can have illegal ranges.

AddIllegalValue

```
bool AddIllegalValue(const std::string& val)
```

Adds an illegal value in string format. This function succeeds and returns `true` if the parameter can have illegal values.

AddLegalRange

```
bool AddLegalRange(const std::string& hi_val,
                   const std::string& low_val)
```

Adds a legal range for the parameter in string format. This function succeeds and returns `true` if the parameter can have legal ranges.

AddLegalValue

```
bool AddLegalValue(const std::string& val)
```

Adds a legal value in string format. This function succeeds and returns `true` if the parameter can have legal values.

AddStringDefault

```
bool AddStringDefault(const std::string& string_default)
```

Adds a default value in string format. This function succeeds and returns `true` if the default value can be set.

AddStringValue

```
bool AddStringValue(const std::string& string_value)
```

Adds a value for the parameter in string format. This function succeeds and returns `true` if the value can be set.

ClearDefaults

```
bool ClearDefaults()
```

Clear all currently set default values. Returns `true` if done successfully.

ClearValues

```
bool ClearValues()
```

Clear all currently set parameter values. Returns `true` if done successfully.

CreateCopy

```
OEParameter* CreateCopy() const
```

Creates a copy of this *OEParameter* instance.

GetAliases

```
OEIterBase<const std::string>* GetAliases() const
```

See *AddAlias* method.

GetBrief

```
std::string GetBrief() const
```

See *SetBrief* method.

GetDetail

```
OEIterBase<const std::string>* GetDetail() const
```

See *AddDetail* method.

GetHasDefault

```
bool GetHasDefault(const unsigned idx = 0) const
```

Returns `true` if a default value for the parameter is set.

GetHasValue

```
bool GetHasValue(const unsigned idx = 0) const
```

Returns `true` if the value of the *OEParameter* object is set.

GetIllegalRanges

```
OEIterBase<const std::pair<std::string, std::string> > * GetIllegalRanges() const
```

See *AddIllegalRange* method.

GetIllegalValues

```
OEIterBase<const std::string>* GetIllegalValues() const
```

See *AddIllegalValue* method.

GetIsList

```
bool GetIsList() const
```

See *SetIsList* method.

GetKeyless

```
unsigned int GetKeyless() const
```

See *SetKeyless* method.

GetLegalRanges

```
OEIterBase<const std::pair<std::string, std::string> > * GetLegalRanges() const
```

Returns the legal ranges in string format.

GetLegalValues

```
OEIterBase<const std::string>* GetLegalValues() const
```

Returns the legal values of the parameter in string format.

GetPrettyLegalValues

```
OEIterBase<const std::string>* GetPrettyLegalValues() const
```

Returns the named legal values of the parameter in string format. This list of legal values may or may-not be idential to the one obtained from *GetLegalValues*, depending on the parameter type.

GetName

```
std::string GetName() const
```

See *SetName* method.

GetOrderPriority

```
int GetOrderPriority() const
```

See *SetOrderPriority* method.

GetRequired

```
bool GetRequired() const
```

See *SetRequired* method.

GetStringDefault

```
std::string GetStringDefault(const unsigned idx = 0) const
```

Returns the string default of the parameter at the specified index. See *AddStringDefault* method.

GetStringDefaults

```
OEIterBase<const std::string>* GetStringDefaults() const
```

See *AddStringDefault* method.

GetStringValue

```
std::string GetStringValue(const unsigned idx = 0) const
```

Returns the string value of the parameter at the specified index. See *AddStringValue* method.

GetStringValues

```
OEIterBase<const std::string>* GetStringValues() const
```

See *AddStringValue* method.

GetVisibility

```
unsigned int GetVisibility() const
```

See *SetVisibility* method.

IsLegalString

```
bool IsLegalString(const std::string& value) const
```

Returns if the specified value in the string format is a legal value. A setting is considered a legal value if it does not match and illegal value or fall within an illegal range (see *OEParameter.AddIllegalValue* and *OEParameter.AddIllegalRange* member functions of this class). Additionally if the parameter has at least one legal value or legal range the setting must match either a legal value or legal range.

IsSet

```
bool IsSet(const unsigned idx = 0) const
```

Returns if the parameter current contains a setting, i.e., a default or an user-defined value.

IsSetToString

```
bool IsSetToString(const std::string& val) const
```

Returns if the parameters current setting (i.e., a default or an user-defined value) matches the specified string value.

Load

```
bool Load()
```

Converts the string defaults and values of this parameter into actual defaults and values. This function will return `false` if the string defaults or values cannot be converted to the actual defaults or values, as well as if the values do not satisfy the specified ranges.

LoadDefault

```
bool LoadDefault()
```

Converts the string defaults of this parameter into actual defaults. This function will return `false` if the string defaults cannot be converted to the actual defaults, as well as if the default values do not satisfy the specified ranges.

LoadValue

```
bool LoadValue()
```

Converts the user-defined string values of this parameter into actual values. This function will return `false` if the string values cannot be converted to the actual values, as well as if the values do not satisfy the specified ranges.

SetBrief

```
bool SetBrief(std::string brief)
```

Sets the brief description of the parameter.

SetIsList

```
bool SetIsList(bool is_list_parameter)
```

Set if the parameter is a list parameter, i.e., if the parameters can value of list of defaults or user-defined values. **Default: False**

SetKeyless

```
bool SetKeyless(const unsigned keyless)
```

Sets the keyless setting of the parameter. To enable a prrameter to be keyless the entry must be a non-zero value. A value of 0 (default) specifies the parameter to be not-keyless. **Default: 0**

SetName

```
bool SetName(std::string name)
```

Sets the name of the parameter, which must be a single word and begin with `'-'`.

SetOrderPriority

```
bool SetOrderPriority(int order)
```

Sets the order priority of the parameter, which is used to control the order parameters appear in help lists. **Default: 0**

SetRequired

```
bool SetRequired(bool required)
```

Sets the flag indicating if the parameter is required (i.e., must be guven a user-defined value). **Default: False**

SetStringDefault

```
bool SetStringDefault(const std::string& string_default)
```

Sets the default value of the parameter as a string. This string default is not converted into the read default until the *OEParameter.LoadDefault* member function is called. **Default: None**

SetStringValue

```
bool SetStringValue(const std::string& string_value)
```

Sets the value of the parameter as a string. This string value is not converted into the real value until the *OEParameter.LoadValue* member function is called. **Default: None**

SetVisibility

```
bool SetVisibility(const unsigned visibility)
```

Sets the visibility of the parameter. Valid settings are defined in the *OEParamVisibility* namespace. ** Default *OEParamVisibility.Simple* **

> **Warning**
>
> This functionality is only available in the C++ toolkits.

OEPredIter

```
template<class P>
class OEPredIter : public OEIterBase<typename P::argument_type>
```

This class represents *OEPredIter*.

The following methods are publicly inherited from *OEIterBase*:

*operator**	*operator-=*	*Copy*
operator++	*operator->*	*Push*
operator+=	*operator A **	*ToFirst*
operator--	*operator bool*	*ToLast*

Constructors

```
OEPredIter()
OEPredIter(const OEPredIter<P>& rhs)
OEPredIter(const P& pred, OEIterBase<A>* rhs)
```

Default and copy constructors.

operator bool

```
operator bool() const
```

Copy

```
OEIterBase<A> *Copy() const
```

Predicate

```
P &Predicate()
```

ToFirst

```
OEIterBase<A> &ToFirst()
```

ToLast

```
OEIterBase<A> &ToLast()
```

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEProtectedBuffer

```
template<class BUFFER>
class OEProtectedBuffer
```

Thread-safe first-in-first-out (FIFO) queue that can have multiple producers and consumers. Meant to wrap around either a *OEBoundedBuffer* or *OEUnboundedBuffer* object to provide thread-safe access to those buffers. The *OEBoundedBuffer* queue should be used to control memory consumption, as the *OEProtectedBuffer.Put* operation will block, allowing the queue to empty first. The *OEUnboundedBuffer* queue will continue to consume memory with every *OEProtectedBuffer.Put* operation.

Constructors

```
OEProtectedBuffer(oesize_t maxsize)
OEProtectedBuffer(oesize_t maxsize, oesize_t threshold)
```

Constructs a new BUFFER, passing `maxsize` to the child buffer implementation. The `threshold` argument is a performance tuning parameter that determines how often *OECondition.Broadcast* is called. Broadcasting to other queue users too often can have negative performance implications as threads can be woken when there is not enough work in the queue to fill up their time quantum. Too infrequently means threads are left idle too long before being awoken with new to perform. The default `threshold` is half of `maxsize`.

Get

```
value_type Get()
```

Return the next item from the queue in a thread-safe manner.

Null

```
value_type Null()
```

The value used to signal termination to the queue. Calling *OEProtectedBuffer.Put* with the value returned by the `Null` method signals that the queue is 'done' and should not accept anymore data. Subsequent *OEProtectedBuffer.Put* operations will fail and return `false`. *OEProtectedBuffer.Get* will continue to return data until the queue is empty, and then start returning the result of the `Null` method as well. Note, this method returns the default constructed value for the `value_type` being stored in the queue, this is typically `0` or `NULL` for integral or pointer types respectively.

Peek

```
value_type Peek()
```

Returns the item that will be returned by the next call to *OEProtectedBuffer.Get*, but does not remove it from the buffer.

> **Warning**
>
> The above is not guaranteed if there are multiple consumers, i.e., other threads calling *OEProtectedBuffer.Get* interleaved with the thread calling *OEProtectedBuffer.Peek*.

Put

```
bool Put(value_type obj)
```

Place an item into the queue in a thread-safe manner.

OERandom

```
class OERandom
```

This class represents *OERandom*.

The *OERandom* class provides a portable pseudo-random number generator.

Constructors

```
OERandom(bool timeseed = false)
```

The 'timeseed' argument to the *OERandom* constructor indicates whether the pseudo random number generator should initialize (or seed) its sequence from the system clock.

GetSeed

```
unsigned int GetSeed() const
```

Return the current state of the random number generator so that a new *OERandom* object can be created and restarted from the given state. To restart a *OERandom* object at a given location, pass this seed into `OERandom.Seed`.

> **Warning**
>
> The datatype of this method will likely change in a future release when a better random number generator will be implemented.

NextFloat

```
float NextFloat()
```

Returns the next pseudo-random floating point value in the range `[0.0, 1.0)`. To convert this value into an integer range, for example `10` discrete values, use `int i = (int)floor(10.0*r.NextFloat())`, which should return values in the range `0` to 9 inclusive.

NextInt

```
int NextInt()
```

Return the next random integer in this sequence of random numbers. The returned integer will never be negative.

Seed

```
void Seed(unsigned int seed)
```

Sets the pseudo random number generator seed to a given value. The *OERandom* class uses a deterministic linear congruential class of pseudo random number generator (PRNG). Setting the seed to a specific value, causes the sequence of subsequent `OERandom.NextFloat` values to be identical on all platforms. This may be useful for testing stochastic algorithms on different architectures/operating systems.

TimeSeed

```
void TimeSeed()
```

Seeds this random number generator based upon the number of seconds since January 1, 1970, Coordinated Universal Time.

OERole

```
class OERole
```

This class represents *OERole*, an object that can be used in a some classification scheme to define a category or role. Each *OERole* is defined by a *role name* string.

Objects inherited from *OERoleSet* can have `OERole` objects added to them as a way of marking them for particular uses.

Constructors

```
OERole()
OERole(const OERole& rhs)
```

Default and copy constructors.

```
OERole(const std::string& name)
```

Standard constructor. It is recommended that classification schemes use hierarchical naming systems, such as `oebio:component:cofactor`, so that they will not conflict with one another when multiple schemes are mixed together.

GetName

```
const char* GetName() const
```

Returns the character string representing the name of the role. An empty character string is returned for a default *OERole* or one constructed with a random unsigned integer.

OERoleSet

```
class OERoleSet
```

This class represents *OERoleSet*, a parent class for objects that need to store *OERole* objects.

An *OERole* is an object that helps define a classification (naming) scheme to represent a category or functional role.

Objects derived from *OERoleSet* can have *OERole* objects added to them as a way of marking them for particular uses.

The following classes derive from this class:

- *OEAtomBondSet*
- *OEMatch*
- *OEMatchBase*

It is possible to test if an *OERole* has been added using the *HasRole* method, or with the *OEHasRole* predicate.

AddRole

```
bool AddRole(const OERole& role)
```

Add the specified *OERole* to this *OERoleSet*. Returns `true` if successful.

```
bool AddRole(const std::string& name)
```

Convenience method for adding a *OERole* by name.

ClearRoles

```
void ClearRoles()
```

Delete all the *OERole* objects from this *OERoleSet*.

DeleteRole

```
bool DeleteRole(const OERole& role)
bool DeleteRole(const std::string& name)
```

Delete the specified *OERole* from this *OERoleSet*. Returns `true` if successful.

GetRoles

```
OEIterBase<const OERole>* GetRoles() const
```

Returns an iterator of every *OERole* object in this *OERoleSet*.

HasRole

```
bool HasRole(const OERole& role) const
```

Returns `true` if the *OERoleSet* contains the specified role.

```
bool HasRole(const std::string& name) const
```

Convenience method for testing membership with a string.

NumRoles

```
unsigned int NumRoles() const
```

Returns the current number of *OERole* objects in this *OERoleSet*.

OERoles

```
class OERoles : public OESystem::OERoleSet
```

The *OERoles* class is a concrete instance of the *OERoleSet* abstract base class. It is designed to be a very simple set container for *OERole* objects. Other containers with more specific purposes include *OEAtomBondSet*, *OEMatch*, and *OEMatchBase*.

The following methods are publicly inherited from *OERoleSet*:

AddRole	*DeleteRole*	*HasRole*
ClearRoles	*GetRoles*	*NumRoles*

Constructors

```
OERoles()
```

Default constructor that generates an empty set.

```
OERoles(const OESystem::OERoleSet& rhs)
```

Copy constructor.

CreateCopy

```
OERoles* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned object is dynamically allocated and owned by the caller.

OEScalarGrid

```
class OEScalarGrid : public OEGrid<float>
```

This class represents *OEScalarGrid*.

An *OEScalarGrid* represents a cubic grid with regular intervals in Cartesian space.

The following methods are publicly inherited from *OEGrid*:

operator=	*SetAll*	*SetXMid*
Clear	*SetDim*	*SetYDim*
CreateCopy	*SetMid*	*SetYMid*
GetDataType	*SetSpacing*	*SetZDim*
IsDataType	*SetXDim*	*SetZMid*

The following methods are publicly inherited from *OEFixedGrid*:

operator()	*GetDataType*	*SetValues*
operator[]	*GetValues*	
CreateCopy	*IsDataType*	

The following methods are publicly inherited from *OEGridBase*:

Clear	*GetXMid*	*GetZMin*
CreateCopy	*GetXMin*	*GridIdxToElement*
ElementToGridIdx	*GetY*	*GridIdxToSpatialCoord*
ElementToSpatialCoord	*GetYDim*	*IsDataType*
GetDataType	*GetYIdx*	*IsInGrid*
GetDim	*GetYInc*	*IsSpacingSet*
GetMid	*GetYMax*	*IsTitleSet*
GetSize	*GetYMid*	*IsXMidSet*
GetSpacing	*GetYMin*	*IsYMidSet*
GetTitle	*GetZ*	*IsZMidSet*
GetX	*GetZDim*	*SetTitle*
GetXDim	*GetZIdx*	*SpatialCoordToElement*
GetXIdx	*GetZInc*	*SpatialCoordToGridIdx*
GetXInc	*GetZMax*	
GetXMax	*GetZMid*	

The following methods are publicly inherited from *OEGridAbstractBase*:

Clear	*GetTitle*	*GetZMid*
GetDataType	*GetXDim*	*IsDataType*
GetDim	*GetXMid*	*IsSpacingSet*
GetMid	*GetYDim*	*SetTitle*
GetSize	*GetYMid*	
GetSpacing	*GetZDim*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OEScalarGrid()
```

Creates a one point *OEScalarGrid* grid object.

```
OEScalarGrid(const OEScalarGrid& rhs)
OEScalarGrid(const OEFixedGrid<float>& rhs)
```

Creates a *OEScalarGrid* object and copies the entire contents of the 'rhs' grid.

```
OEScalarGrid(float minmax[6], float spacing, const char* title = "")
OEScalarGrid(double minmax[6], double spacing,
             const char* title = "")
```

Creates a *OEScalarGrid* object from a minimum bounding box ('minmax'). The size of each length between grid points is determined by the 'spacing' parameter. The 'title' parameter is provided as a convenience for setting the title upon construction. It can later be retrieved with the *OEGridBase.GetTitle* method.

```
OEScalarGrid(int xdim, int ydim, int zdim, float midx, float midy,
             float midz, float spacing, std::string title = "",
             float* dataptr = 0)
```

Creates and initializes a *OEScalarGrid* object. The grid data can be copied from the optional parameter 'dataptr'. The pointer should be an array of size `xdim*ydim*zdim*sizeof(float)`.

Clear

```
void Clear()
```

Reallocates memory for the grid, effectively clearing all the grid values. Calls through to *OEBase.Clear*.

CreateCopy

```
OEBase* CreateCopy() const
```

Makes a copy of the grid instance.

GetDataType

```
const void* GetDataType() const
```

Returns the data type of the grid.

GetType

```
unsigned int GetType() const
```

Returns the data type tag for the grid.

IsDataType

```
bool IsDataType(const void* type) const
```

Returns whether type is the same as the instance this method is called on (*i.e.* *OEScalarGrid*).

SetType

```
bool SetType(unsigned int t)
```

Sets the tag associated with this grid.

OESharedPtr

```
template<class T>
class OESharedPtr
```

Warning

This class has been deprecated. Use `std::shared_ptr` instead. C++ only class. This class is not visible in any other languages.

This class is a classic smart pointer used for automatically managing the dynamic memory allocated for the object `T`. Explicitly calling `delete` is generally frowned upon in modern C++ due to the propensity of explicit `delete` leading to resource leaks and crashes. Deleting memory should be handled more automatically by mechanisms like *OEOwnedPtr* and *OESharedPtr*.

Hint

One of the most important differences to understand between *OEOwnedPtr* and *OESharedPtr* is that *OESharedPtr* is always usable inside STL containers like `std::vector<OESharedPtr<T> >`. *OEOwnedPtr* can only be used

inside STL containers if the compiler supports C++11 move constructors. Once all common compilers support C++11, the use of *OESharedPtr* should be replaced with *OEOwnedPtr* in many cases.

OESharedPtr is designed to mimic the standard C++ std::shared_ptr in functionality with some important differences.

The first and most obvious difference is that *OESharedPtr* will compile and work in all compilers *OpenEye* supports without having to juggle whether it is coming from the `std` or `std::tr1` namespaces.

The second key difference between *OESharedPtr* and std::shared_ptr is whether `const` is maintained when accessing the object being pointed to. By default, in order to maintain compatibility with C, C++ allows non-const access to const pointers. By extension, this same rule applied to C++ smart pointers. This leads to very surprising behavior like the following:

```
struct Foo
{
  int _x;
  void AlterFoo() { _x = 1; }
};

struct BarOwnsFoo
{
  Foo *_foo;
  void AConstMethod() const { _foo->AlterFoo(); }
};
```

Even though `AConstMethod` is `const`, it isn't actually logically const. The internals of the class are changing. This behavior may seem benign at first, but it has vast implications in a library seeking to be thread-safe. The `const` on a method implies that the method is thread-safe, especially in C++11. In order to enforce thread-safety at runtime, *OESharedPtr* maintains the "const'ness" of its accessor methods. Therefore, the following is actually a compile error:

```
struct Foo
{
  int _x;
  void AlterFoo() { _x = 1; }
};

struct BarOwnsFoo
{
  OESharedPtr<Foo> _foo;
  void AConstMethod() const { _foo->AlterFoo(); }
};
```

Catching thread-safety problems very early, at compilation, is highly desirable for *OpenEye*. Hence the reason to differ from the standard smart pointer semantics in this way.

Constructors

```
OESharedPtr()
OESharedPtr(T* ptr)
```

Default constructors the pointer to `NULL`. Or constructs a smart pointer around the pointer `ptr`.

Get

```
T* Get()
const T* Get() const
```

Access the pointer held by this smart pointer while maintaining `const` correctness.

OESkewGrid

```
class OESkewGrid : public OEGridAbstractBase
```

This class represents *OESkewGrid*.

The following methods are publicly inherited from *OEGridAbstractBase*:

Clear	*GetTitle*	*GetZMid*
GetDataType	*GetXDim*	*IsDataType*
GetDim	*GetXMid*	*IsSpacingSet*
GetMid	*GetYDim*	*SetTitle*
GetSize	*GetYMid*	
GetSpacing	*GetZDim*	

The following methods are publicly inherited from *OEBase*:

operator=	*GetData*	*IsDataType*
operator+=	*GetDataIter*	*SetBaseData*
AddBaseData	*GetDataType*	*SetBoolData*
AddData	*GetDoubleData*	*SetData*
Clear	*GetFloatData*	*SetDoubleData*
CreateCopy	*GetIntData*	*SetFloatData*
DeleteData	*GetStringData*	*SetIntData*
GetBoolData	*HasData*	*SetStringData*

Constructors

```
OESkewGrid()
```

Creates a one point *OESkewGrid* grid object.

```
OESkewGrid(const OEScalarGrid& rhs)
OESkewGrid(const OESkewGrid& rhs)
OESkewGrid(const OESkewGridImpl& rhs)
OESkewGrid(const OEFixedGrid<float>& rhs)
```

Creates a *OESkewGrid* object and copies the entire contents of the 'rhs'.

operator bool

```
operator bool() const
```

AsRegularGrid

```
const OEFixedGrid<float>& AsRegularGrid() const
```

Clear

```
void Clear()
```

Reallocates memory for the grid, effectively clearing all the grid values. Calls through to *OEBase.Clear*.

CreateCopy

```
OESystem::OEBase* CreateCopy() const
```

Makes a copy of the grid instance.

ElementToGridIdx

```
bool ElementToGridIdx(unsigned int el, unsigned int& ix,
                      unsigned int& iy, unsigned int& iz) const
```

The element 'el' is the one dimensional index into the grid points. This function converts this one dimensional index into the three dimensional indices (ix, iy, iz) for the associated grid point. Returns `false` if 'el' is outside the array bounds.

ElementToSpatialCoord

```
bool ElementToSpatialCoord(unsigned int el, float& x, float& y,
                           float& z) const
```

The element 'el' is the one dimensional index into the grid points. This function converts this one dimensional index into the Cartesian coordinates (x,y,z) of the associated grid point. Returns false if 'el' is outside the array bounds.

GetDataType

```
const void* GetDataType() const
```

Returns the data type of the grid.

GetDim

```
bool GetDim(unsigned int &xdim, unsigned int &ydim, unsigned int &zdim) const
```

Retrieves the grid dimensions (*i.e.* the wideness of the grid for a specific axis) into 'xdim', 'ydim', 'zdim' arguments.

GetMid

```
bool GetMid(float &midx, float &midy, float &midz) const
```

Retrieves the grid midpoint (*i.e.* the center of the grid) into 'midx', 'midy' and 'midz' arguments. The midpoint defines where the grid is located in Cartesian space. Returns `false` if the midpoint is not completely defined (! *IsXMidSet()* || ! *IsYMidSet()* || ! *IsZMidSet()*)

GetReentrant

```
unsigned int GetReentrant() const
```

GetSize

```
unsigned int GetSize() const
```

Returns the number of grid points inside this grid.

GetSpaceGroup

```
unsigned int GetSpaceGroup() const
```

GetSpacing

```
float GetSpacing() const
```

Returns the length between grid points.

GetTitle

```
std::string GetTitle() const
```

Returns the title of this grid. The title defaults to an empty string.

GetType

```
unsigned int GetType() const
```

Returns the data type tag for the grid.

GetUnitCell

```
bool GetUnitCell(float& A, float& B, float& C, float& alpha,
                 float& beta, float& gamma) const
```

GetValues

```
float *GetValues()
const float *GetValues() const
```

GetXDim

```
unsigned int GetXDim() const
```

Returns how many grid points there are for the x-dimension of the grid.

GetXMid

```
float GetXMid() const
```

Returns the center of this grid along the x-axis.

GetYDim

```
unsigned int GetYDim() const
```

Returns how many grid points there are for the y-dimension of the grid.

GetYMid

```
float GetYMid() const
```

Returns the center of this grid along the y-axis.

GetZDim

```
unsigned int GetZDim() const
```

Returns how many grid points there are for the z-dimension of the grid.

GetZMid

```
float GetZMid() const
```

Returns the center of this grid along the z-axis.

HasSpaceGroup

```
bool HasSpaceGroup() const
```

HasUnitCell

```
bool HasUnitCell() const
```

IsDataType

```
bool IsDataType(const void* type) const
```

Returns whether type is the same as the instance this method is called on (*i.e.* *OESkewGrid*).

IsRegular

```
bool IsRegular() const
```

IsSpacingSet

```
bool IsSpacingSet() const
```

Returns whether the spacing for this grid has been specified.

IsTitleSet

```
bool IsTitleSet() const
```

Returns whether there is a title associated with this grid.

SetDim

```
bool SetDim(unsigned int xdim, unsigned int ydim, unsigned int zdim)
```

Sets the number of grid points for each dimension.

SetMid

```
bool SetMid(float midx, float midy, float midz)
```

Sets the mid-point (center) of the grid.

SetReentrant

```
void SetReentrant(bool state = true)
```

SetSpaceGroup

```
bool SetSpaceGroup(unsigned int group)
```

SetSpacing

```
bool SetSpacing(float spacing)
```

Sets the length between grid points.

SetTitle

```
bool SetTitle(const std::string& title)
```

Sets the title associated with this grid.

SetType

```
void SetType(unsigned int type)
```

Sets the tag associated with this grid.

SetUnitCell

```
bool SetUnitCell(float A, float B, float C, float alpha,
                 float beta, float gamma, unsigned int nx,
                 unsigned int ny, unsigned int nz)
```

SetValues

```
bool SetValues(const float* values, unsigned int length)
```

OEStopwatch

```
class OEStopwatch
```

This class represents *OEStopwatch*.

The *OEStopwatch* class provides a portable mechanism for measuring elapsed CPU time of a process.

Constructors

```
OEStopwatch()
```

Default constructor.

Elapsed

```
float Elapsed()
```

Returns the number of elapsed CPU seconds (as a floating point value) since *OEStopwatch.Start* was last called. This method is identical to *OEStopwatch.Lap*. This method does not update the internal 'start' time.

Lap

```
float Lap()
```

Returns the number of elapsed CPU seconds (as a floating point value) since *OEStopwatch.Start* was last called. This method is identical to *OEStopwatch.Elapsed*. This method does not update the internal 'start' time.

Start

```
void Start()
```

Sets the internal 'start' time of the *OEStopwatch* to the current time. This method should be called before calling either *OEStopwatch.Elapsed* or *OEStopwatch.Lap*. This method may be called more than once to reinitialize the 'start' time to the current time.

OEThreadedDots

```
class OEThreadedDots
```

This class provides a threadsafe version of *OEDots*. Multiple threads can call *OEThreadedDots.Update* to update the progress in an efficient manner in a thread-safe way. This class will create an internal operating system thread

to handle output to *OEPlatform.oeerr*. The complete progress is not guaranteed to be output to the terminal until *OEThreadedDots.Total* is called.

> **Warning**
>
> It is an error to call *OEThreadedDots.Update* after calling *OEThreadedDots.Total*. The object is essentially shut down and new one should be created.

Constructors

```
OEThreadedDots(unsigned int bigstep, unsigned int smallstep,
               const char* objectname = "")
```

Controls how the following output gets updated:

```
.......... 10000 things processed.
.......... 20000 things processed.
.......... 30000 things processed.
.......... 40000 things processed.
.......... 50000 things processed.
.......... 60000 things processed.
.......... 70000 things processed.
.......... 80000 things processed.
```

bigstep

The number of elements before a new line is generated. The above example is a `bigstep` of `10000`.

smallstep

The number of elements before a single new dot, `.`, is printed. The above example is a `smallstep` of `1000`. Or 10 dots per line.

objectname

A description of what is being processed. In the above example, this is "things".

Total

```
void Total()
```

Prints a final summary line like the following:

```
Total: 80000 things processed.
```

> **Note**
>
> Calling this method waits for all information to flush to the IO thread and then terminates and joins that thread. So this function is effectively a blocking call on those operations. But the operations are typically very cheap, so the blocking nature is not noticeable.

Update

```
void Update(unsigned int step = 1)
```

Update the progress with the number of elements specified by `step`. The default is 1 element, but any number can be specified. However, `0` is not a valid step size and will throw an error.

> **Note**
>
> For *OEThreadedDots*, this method is thread-safe to call from multiple threads even though it is a non-const method.

OETracerBase

```
class OETracerBase
```

OETracerBase is an abstract class which defines the interface necessary for setting up and using a tracer to report progress of task(s).

The following class derives from this class:

- *OEConsoleProgressTracer*

Aborted

```
bool Aborted()=0
```

Pop

```
void Pop()=0
```

Push

```
void Push(std::string task)=0
```

Refresh

```
void Refresh()=0
```

SetProgress

```
void SetProgress(double percent)=0
void SetProgress(unsigned int current, unsigned int total)=0
```

This pure virtual method should report the current progress of the task when it has reached the percentage of completion specified by `percent` or by `current` / `total`.

SetTask

```
void SetTask(std::string task)=0
```

This pure virtual method should register the tracer with a task name specified by, `task`. It should start the timer to trace the progress of the task.

OETypedParameter

```
template<class TT>
class OETypedParameter : public OEParameter
```

The *OETypedParameter* is an abstract base class. This is a templated class that provides type-specific implementation of parameters.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

The *OETypedParameter* class defines the following public methods:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

The following classes derive from this class:

- *OEPODParameter*

AddDefault

```
bool AddDefault(const TT& dflt)
```

Add the specified value as a default for the parameter.

AddValue

```
bool AddValue(const TT& value)
```

Add the specified value as a user-defined value for the parameter.

GetDefault

```
bool GetDefault(TT& dflt, const unsigned idx = 0) const
```

Returns the default value of the parameter at the specified index, if available. See *AddDefault* method.

GetDefaults

```
OEIterBase<const TT>* GetDefaults() const = 0
```

See *AddDefault* method.

GetSetting

```
bool GetSetting(TT& setting, unsigned int idx = 0) const
```

Returns the current settings of the parameter at the specified index, if available.

GetSettings

```
OEIterBase<const TT>* GetSettings() const
```

Returns all the current settings of the parameter.

GetValue

```
bool GetValue(TT& value, const unsigned idx = 0) const
```

Returns the user defined value of the parameter at the specified index, if available. See *AddValue* method.

GetValues

```
OEIterBase<const TT>* GetValues() const = 0
```

See *AddValue* method.

SetDefault

```
bool SetDefault(const TT& dflt)
```

Sets the parameter default value. Overrides any existing defaults. See *AddDefault* method.

SetValue

```
bool SetValue(const TT& value)
```

Sets the parameter user-defined value. Overrides any existing user-defined value. See *AddDefault* method.

OEUnaryFalse

```
template<class T>
class OEUnaryFalse : public OEIsFalse<T>
```

This class is just a synonym for *OEIsFalse*.

The following methods are publicly inherited from *OEIsFalse*:

operator()	*CreateCopy*

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

OEUnaryFunction

```
template<class Arg, class Result, bool ConstArg, bool ConstFcn>
class OEUnaryFunction
```

This class represents *OEUnaryFunction*.

This is the default functor base-class in *OEChem*.

The following specializations exist for this template:

- OESystem::OEUnaryFunction< Arg, Result, false, false >
- OESystem::OEUnaryFunction< Arg, Result, false, true >
- OESystem::OEUnaryFunction< Arg, Result, true, false >

The following classes derive from this class:

- *OEHasAlternateLocation*
- *OEHasPDBAtomIndex*
- *OEIsAlphaCarbon*

- *OEIsBackboneAtom*
- *OEHasIdx*
- *OEIdxSelected*
- *OEIsMember*
- *OEIsMemberPtr*
- *OEMatch*
- *OEAtomIsInAgent*
- *OEAtomIsInProduct*
- *OEAtomIsInReactant*
- *OEAtomIsInResidue*
- *OEAtomIsInRing*
- *OEBondIsInRing*
- *OEGetNbrAtom*
- *OEHasAtomName*
- *OEHasAtomStereoSpecified*
- *OEHasAtomicNum*
- *OEHasBondStereoSpecified*
- *OEHasChainID*
- *OEHasFragmentNumber*
- *OEHasIdx*
- *OEHasMapIdx*
- *OEHasOrder*
- *OEHasResidueNumber*
- *OEIsAromaticAtom*
- *OEIsAromaticBond*
- *OEIsCAlpha*
- *OEIsCarbon*
- *OEIsChiralAtom*
- *OEIsChiralBond*
- *OEIsHalide*
- *OEIsHalogen*
- *OEIsHeavy*
- *OEIsHydrogen*
- *OEIsMember*
- *OEIsMemberPtr*
- *OEIsNitrogen*

- *OEIsOxygen*
- *OEIsPhosphorus*
- *OEIsPolar*
- *OEIsPolarHydrogen*
- *OEIsRGroup*
- *OEIsRotor*
- *OEIsSulfur*
- *OEIsValidAtomValence*
- *OEIsValidMDLAtomValence*
- *OEMatchFunc*
- *OENthAtom*
- *OEPartPredT*
- *OEAnd*
- *OENot*
- *OEOr*
- *OEUnaryFalse*
- *OEUnaryPredicate*

operator()

```
Result operator()(const Arg &) const = 0
```

It is a virtual const *OEUnaryFunction.operator()* which takes a const argument.

CreateCopy

```
base_type* CreateCopy() const = 0
```

It is a virtual const constructor which allows copying of concrete derived objects using a reference to this base class.

OEUnaryPredicate

```
template<class Arg>
class OEUnaryPredicate : public OEUnaryFunction<Arg, bool>
```

This class represents *OEUnaryPredicate*.

This abstract base-class defines predicates in *OEChem*. Predicates are a special class of functors which, by convention, has several restriction. These restrictions make predicates predictable and reliable. To the degree possible, these restrictions are enforced by the API. *OEUnaryPredicate.CreateCopy* is a virtual constructor which allows copying of concrete derived objects using a reference to this base class.

By definition, a predicates is a functor which always return a boolean. By convention, the *OEUnaryPredicate.operator()* of a predicate does not modify the object it takes as an argument and itself is not modified by the evaluation. These conventions guarantee that the result of a series of predicate evaluations will not be order dependent.

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	CreateCopy

The following atom predicate classes classes derive from this class:

- *OEAtomIsInAgent*
- *OEAtomIsInChain*
- *OEAtomIsInProduct*
- *OEAtomIsInReactant*
- *OEAtomIsInResidue*
- *OEAtomIsInRing*
- *OEAtomMatchResidue*
- *OEHasAlphaBetaUnsat*
- *OEHasAlternateLocation*
- *OEHasAnisou*
- *OEHasAtomicNum*
- *OEHasAtomName*
- *OEHasAtomStereoSpecified*
- *OEHasChainID*
- *OEHasFormalCharge*
- *OEHasFragmentNumber*
- *OEHasHvyDegree*
- *OEHasIdx*
- *OEHasMapIdx*
- *OEHasPDBAtomIndex*
- *OEHasResidueNumber*
- *OEHasSecondaryStructure*
- *OEIsAlphaCarbon*
- *OEIsAromaticAtom*
- *OEIsAtomHybridization*
- *OEIsBackboneAtom*
- *OEIsCAlpha*
- *OEIsCarbon*
- *OEIsChiralAtom*
- OEIsColorAtomPred
- *OEIsHalide*
- *OEIsHalogen*
- *OEIsHeavy*

- *OEIsHetero*
- *OEIsHydrogen*
- *OEIsInvertibleNitrogen*
- *OEIsMetal*
- *OEIsNitrogen*
- *OEIsNonRingAtomDoubleBondedToRing*
- *OEIsNucleicAcidBase*
- *OEIsNucleicAcidPhosphate*
- *OEIsNucleicAcidSugar*
- *OEIsOxygen*
- *OEIsPhosphorus*
- *OEIsPolar*
- *OEIsPolarHydrogen*
- *OEIsRGroup*
- *OEIsSulfur*
- *OEIsValidAtomValence*
- *OEIsValidMDLAtomValence*
- *OEIsWater*

The following bond predicate classes derive from this class:

- *OEBondIsInChain*
- *OEBondIsInRing*
- *OEHasBondStereoSpecified*
- *OEHasOrder*
- *OEIsAromaticBond*
- *OEIsChiralBond*
- *OEIsRotor*

The following template predicate classes derive from this class:

- *OEIdxSelected*
- *OEIsMember*
- *OEIsMemberPtr*
- *OEIsMember*
- *OEIsMemberPtr*
- *OEMatchFunc*
- *OENthAtom*
- *OEPartPredT*
- *IsFalse*

- *IsTrue*
- *OEAnd*
- *OENot*
- *OEOr*
- *OEUnaryFalse*

The following specializations exist for this template:

OEUnaryBasePred	OESystem::OEUnaryPredicate< OESystem::OEBase >
OEUnaryAtomPred	OESystem::OEUnaryPredicate< OEChem::OEAtomBase >
OEUnaryBondPred	OESystem::OEUnaryPredicate< OEChem::OEBondBase >
OEUnaryConfPred	OESystem::OEUnaryPredicate< OEChem::OEConfBase >
OEUnaryMolBasePred	OESystem::OEUnaryPredicate< OEChem::OEMolBase >
OEUnaryGraphMolPred	OESystem::OEUnaryPredicate< OEChem::OEGraphMol >

operator()

```
bool operator()(const Arg& a) const = 0
```

CreateCopy

```
base_type* CreateCopy() const = 0
```

CreatePredicateCopy

```
OEFORCEINLINE OEUnaryPredicate<Arg> * CreatePredicateCopy() const
```

OEUnaryTrue

```
template<class T> class OEUnaryTrue : public OEIsTrue<T>
```

This class is just a synonym for *OEIsTrue*.

The following methods are publicly inherited from *OEIsTrue*:

operator()	*CreateCopy*

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEUnboundedBuffer

```
template<class T>
class OEUnboundedBuffer
```

First-in-first-out (FIFO) queue that only allows for any number of items to be stored in the buffer, hence, "unbounded". It is not inherently thread safe by itself. Use this in conjunction with a *OEProtectedBuffer* to make a thread-safe queue.

Constructors

```
OEUnboundedBuffer(oesize_t)
```

The constructor argument is unused and reserved for future use. It is only there to comply with the *OEBoundedBuffer* constructor interface.

Empty

```
bool Empty() const
```

Returns whether the queue contains any items.

Full

```
bool Full() const
```

Returns `true` if the queue contains `maxsize` items.

Get

```
T Get()
```

Pops the next item off of the queue.

Peek

```
T Peek()
```

Returns the item that will be returned by the next call to *OEBoundedBuffer.Get*, but does not remove it from the queue.

Put

```
void Put(T obj)
```

Pushes an item into the queue.

Size

```
oesize_t Size() const
```

Returns the current number of items in the queue.

OEUnownedPtr

```
template<class T>
class OEUnownedPtr
```

> **Warning**
>
> This class has been deprecated. Use raw pointers instead. C++ only class. This class is not visible in any other languages.

This class is designed to provide the same "const correct-ness" guarantees that *OEOwnedPtr* provides. A description of this "const correct-ness" can be found in the *OEOwnedPtr* section. This class is really only useful to ensure pointer dereferencing does not inadvertently create a thread safety problem.

Constructors

```
OEUnownedPtr()
OEUnownedPtr(T* ptr)
```

Default constructors the pointer to `NULL`. Or constructor a smart pointer around the pointer `ptr`.

Get

```
T* Get()
const T* Get() const
```

Access the pointer held by this smart pointer while maintaining `const` correctness.

Release

```
T* Release()
```

Returns the pointer owned by this smart pointer. This object will cease to point to the object in question. The object will effectively become a `NULL` pointer.

> **Warning**
>
> No memory management is performed on the pointer with `delete`.

OEWallTimer

```
class OEWallTimer
```

This class represents *OEWallTimer*.

Constructors

```
OEWallTimer()
```

Default and copy constructors.

Elapsed

```
float Elapsed()
```

Lap

```
float Lap()
```

Start

```
void Start()
```

oeprogressstream

```
class oeprogressstream : public OEPlatform::oeiwrapperstream
```

This class represents *oeprogressstream*.

The following methods are publicly inherited from *oeiwrapperstream*:

clear	*set*	*stream*

The following methods are publicly inherited from *oeistream*:

operator>>	*gettoken*	*skipbyte*
close	*open*	*skipline*
get	*peekbyte*	
getbyte	*read*	
getline	*skip*	

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeprogressstream(OEPlatform::oeistream* istream, bool ownstream,
                 OETracerBase* tracer = new OEDefaultProgressTracer,
                 bool owntracer = true)
```

Default and copy constructors.

SetTracer

```
void SetTracer(OETracerBase* tracer, bool owntracer = true)
```

4.3.2 OESystem Constants

OEAnnotationPrimitive

This namespace contains constants.

Container

Sphere

Ellipsoid

Box

Cylinder

Arrow

Line

Surface

Spline

SplineSurface

Torus

Text

Symbol

OEAnnotationRenderHint

This namespace contains constants.

Solid

Mesh

Points

OEErrorLevel

This namespace contains constants.

Undefined

Debug

Verbose

Info

Warning

Error

Fatal

Usage

Quiet

MAX

MIN

Default

The default error level is *OEErrorLevel.Info*.

OEGridFileType

This namespace contains constants.

UNDEFINED

Grasp

GRD

CCP4

XPLOR

Ascii

OEGridMaskType

This namespace contains constants.

ValuePlus

ValueMinus

GridPlus

GridMinus

Gaussian

GaussianMinus

GaussianSmooth

GaussianSmoothMinus

Default

OEGridType

This namespace contains constants.

GenericGrid

ElectrostaticGrid

FredGrid

RegularMap

DifferenceMap

ETGrid

EonGrid

MaxGridType

OEMemPoolMode

This namespace contains constants.

SingleThreaded

Mutexed

Spinlocked

ThreadLocal

BoundedCache

UnboundedCache

System

Default

OEMTZMapType

This namespace contains constants.

See also

- *OEReadMTZ* function
- *OEMTZMapTypeToString* function
- *OEMTZMapTypeFromString* function

Fc

Fdelwt

FiveFoThreeFc

Fo

FoFc

FoFcSigmaA

FoFom

FoSquared

Fwt

None

ThreeFoTwoFc

TwoFoFc

TwoFoFcSigmaA

User

UserDiff

OERecordReturnCode

> **Attention**
>
> This API is currently available in C++ and Python.

The *OERecordReturnCode* returns codes that convey status regarding fields in a *OERecord*.

This namespace contains the following constants:

OK

Returned if the desired field exists.

NoValue

Returned if the field does not have a value.

NullValue

Returned if the field has a null value.

TypeMismatch

Returned if the field has a different type than the one specified in the call.

MetadataMismatch

Returned if the field being queried has different metadata than the metadata specified in the call.

OESystem

This namespace contains constants.

OEBlack



OEBlack (R=0, G=0, B=0)

OEBlue



OEBlue (R=0, G=0, B=255)

OEBlueTint



OEBlueTint (R=175, G=214, B=255)

OEBrass



OEBrass (R=220, G=175, B=90)

OEBrown



OEBrown (R=175, G=117, B=89)

OECopper



OECopper (R=245, G=175, B=145)

OECyan



OECyan (R=0, G=255, B=255)

OEDarkBlue



OEDarkBlue (R=0, G=0, B=128)

OEDarkBrown



OEDarkBrown (R=102, G=0, B=0)

OEDarkCyan



OEDarkCyan (R=0, G=128, B=128)

OEDarkGreen



OEDarkGreen (R=0, G=128, B=0)

OEDarkGrey



OEDarkGrey (R=64, G=64, B=64)

OEDarkMagenta



OEDarkMagenta (R=128, G=0, B=128)

OEDarkOrange



OEDarkOrange (R=170, G=85, B=0)

OEDarkPurple



OEDarkPurple (R=94, G=0, B=94)

OEDarkRed



OEDarkRed (R=170, G=0, B=0)

OEDarkRose



OEDarkRose (R=160, G=0, B=66)

OEDarkSalmon



OEDarkSalmon (R=255, G=76, B=76)

OEDarkYellow



OEDarkYellow (R=128, G=128, B=0)

OEGold



OEGold (R=255, G=220, B=70)

OEGreen



OEGreen (R=0, G=255, B=0)

OEGreenBlue



OEGreenBlue (R=46, G=139, B=87)

OEGreenTint



OEGreenTint (R=152, G=255, B=179)

OEGrey



OEGrey (R=180, G=180, B=180)

OEHotPink



OEHotPink (R=255, G=0, B=101)

OELightBlue



OELightBlue (R=160, G=160, B=255)

OELightBrown



OELightBrown (R=184, G=160, B=66)

OELightGreen



OELightGreen (R=160, G=255, B=160)

OELightGrey



OELightGrey (R=191, G=191, B=191)

OELightOrange



OELightOrange (R=255, G=140, B=75)

OELightPurple



OELightPurple (R=255, G=140, B=255)

OELightSalmon



OELightSalmon (R=255, G=124, B=112)

OELimeGreen



OELimeGreen (R=170, G=255, B=0)

OEMagenta



OEMagenta (R=192, G=0, B=192)

OEMandarin



OEMandarin (R=255, G=156, B=0)

OEMediumBlue



OEMediumBlue (R=112, G=112, B=255)

OEMediumBrown



OEMediumBrown (R=140, G=112, B=76)

OEMediumGreen



OEMediumGreen (R=140, G=255, B=140)

OEMediumOrange



OEMediumOrange (R=255, G=112, B=66)

OEMediumPurple



OEMediumPurple (R=255, G=0, B=255)

OEMediumSalmon



OEMediumSalmon (R=255, G=112, B=112)

OEMediumYellow



OEMediumYellow (R=255, G=255, B=112)

OEOliveBrown



OEOliveBrown (R=79, G=70, B=0)

OEOliveGreen



OEOliveGreen (R=69, G=94, B=69)

OEOliveGrey



OEOliveGrey (R=83, G=76, B=66)

OEOrange



OEOrange (R=255, G=128, B=0)

OEPewter



OEPewter (R=140, G=140, B=135)

OEPink



OEPink (R=255, G=0, B=128)

OEPinkTint



OEPinkTint (R=255, G=171, B=187)

OEPurple



OEPurple (R=170, G=0, B=255)

OERed



OERed (R=255, G=0, B=0)

OERedOrange



OERedOrange (R=255, G=69, B=0)

OERoyalBlue



OERoyalBlue (R=71, G=71, B=184)

OESeaGreen



OESeaGreen (R=0, G=250, B=109)

OESilver

 **OESilver (R=240, G=240, B=235)**

OESkyBlue

 **OESkyBlue (R=58, G=144, B=255)**

OETransparentColor

The color that can be used to generate images with transparent background.

See also

- Molecule Depiction with Transparent Background section in the **OEDepict TK** manual

OEViolet

 **OEViolet (R=238, G=130, B=238)**

OEWhite

 **OEWhite (R=255, G=255, B=255)**

OEYellow

 **OEYellow (R=255, G=255, B=0)**

OEYellowTint

 **OEYellowTint (R=246, G=246, B=117)**

OEThrow

This is the globally available error handler that is used to report all diagnostics generated by the *OEChem*, *OESystem* and *OEPlatform* libraries. The behavior of `OEThrow` may be customized via the various methods available to *OEErrorHandler*. For example, warning and error messages may be redirected to a file or string stream, instead of to `OEPlatform.oeerr` (the default).

Example

OEParamVisibility

This namespace contains constants.

Simple

Normal

Hidden

MAX

4.3.3 OESystem Functions

OEAddContour

```
void OEAddContour(OEGridAbstractBase& grid, float level,
                  const OEColor& c)
```

Adds a grid contour at 'level' to the 'grid' (*OEGridAbstractBase*) with the color 'c'.

Note

Contours are only persistent if written to OEBinary (`.oeb`).

See also

- *OEGetContourLevel*
- *OEGetContourColor*
- *OENumContours*

OEAddGrid

```
template<class GridType1, class GridType2>
bool OEAddGrid(GridType1& grid1, const GridType2& grid2)
```

OEAnnotateArrow

```
OEAnnotation OEAnnotateArrow(const char* title, float* A, float* B,
                             float percentCylinder, float r1,
                             float r2, float r3)
```

OEAnnotateBox

```
OEAnnotation OEAnnotateBox(const char* title, float* center,
                           float* extents)
```

OEAnnotateCylinder

```
OEAnnotation OEAnnotateCylinder(const char* title, float* A, float* B,
                                float r1, float r2)
```

OEAnnotateEllipsoid

```
OEAnnotation OEAnnotateEllipsoid(const char* title, float* center,
                                 float a, float b, float c,
                                 float* dir1, float* dir2, float* dir3)
```

OEAnnotateLine

```
OEAnnotation OEAnnotateLine(const char* title, float* A, float* B)
```

OEAnnotateSphere

```
OEAnnotation OEAnnotateSphere(const char* title, float* center,
                              float radius,
                              float* hemisphereDirection = 0)
```

OEAnnotateSpline

```
OEAnnotation OEAnnotateSpline(const char* title, float* coords,
                              unsigned int numPoints,
                              unsigned int splineResolution)
```

OEAnnotateSplineSurface

```
OEAnnotation OEAnnotateSplineSurface(const char* title, float* coords,
                                     unsigned int numPoints,
                                     unsigned int splineResolution,
                                     int segmentResolution,
                                     int crossResolution, float width,
                                     float thickness, int splineType,
                                     int style)
```

OEAnnotateSymbol

```
OEAnnotation OEAnnotateSymbol(const char* title, float* A,
                              const char* symbol)
```

OEAnnotateText

```
OEAnnotation OEAnnotateText(const char* title, float* A,
                            const char* text)
```

OEAnnotateTorus

```
OEAnnotation OEAnnotateTorus(const char* title, float* A, float r1,
                             float r2, float* direction)
```

OEAttachData

OEBDataLengthBytes

```
unsigned int OEBDataLengthBytes(OEULongLong)
unsigned int OEBDataLengthBytes(unsigned int)
```

OEBFPosTEndian

```
unsigned char OEBFPosTEndian()
bool OEBFPosTEndian(unsigned int)
```

The first overload returns a character representing the endian-ness of a `oefpos_t` so that `oefpos_t` data can be directly written to a file. The second overload interprets this character and returns `true` if the data will need to be swapped to match the current machine endian-ness.

OEBFloatEndian

```
unsigned char OEBFloatEndian()
bool OEBFloatEndian(unsigned int)
```

The first overload returns a character representing the endian-ness of a `float` or `double` so that `float` or `double` data can be directly written to a file. The second overload interprets this character and returns `true` if the data will need to be swapped to match the current machine endian-ness.

OEBIntEndian

```
unsigned char OEBIntEndian()
bool OEBIntEndian(unsigned int)
```

The first overload returns a character representing the endian-ness of a `int` or `unsigned int` so that `int` or `unsigned int` data can be directly written to a file. The second overload interprets this character and returns `true` if the data will need to be swapped to match the current machine endian-ness.

OEBPeekTag

```
bool OEBPeekTag(OEPlatform::oeistream& ifs, char* tag)
bool OEBPeekTag(OEPlatform::oeistream& ifs, char* tag,
                unsigned char& tagtype)
```

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEBReadData

```
template<class T>
bool OEBReadData(OEPlatform::oeistream& ifs, T& t, bool swab)
template<class T>
bool OEBReadData(OEPlatform::oeistream& ifs, T t, size_t nbytes,
                 bool swab)
```

OEBReadDataLength

```
OEULongLong OEBReadDataLength(OEPlatform::oeistream &,
                              OEULongLong maxLen = UINT_MAX)
```

Reads a variable length encoding integer from a binary file.

OEBReadOEBase

```
unsigned int OEBReadOEBase(OEPlatform::oeistream& ifs, OEBase& base,
                           OEULongLong len,
                           const OEBinaryIOHandlerBase& handler)
```

OEBReadPOD

```
unsigned int OEBReadPOD(OEPlatform::oeistream& ifs, OEBase& b,
                        OEULongLong len)
```

OEBReadTag

```
bool OEBReadTag(OEPlatform::oeistream& ifs, char* tag)
bool OEBReadTag(OEPlatform::oeistream& ifs, char* tag,
                unsigned char& tagtype)
```

OEBShortEndian

```
unsigned char OEBShortEndian()
bool OEBShortEndian(unsigned int)
```

The first overload returns a character representing the endian-ness of a `short` so that `short` data can be directly written to a file. The second overload interprets this character and returns `true` if the data will need to be swapped to match the current machine endian-ness.

OEBWriteData

```
template<class T>
bool OEBWriteData(OEPlatform::oeostream& ofs, const T& t)
```

Write the binary representation of `t` to the file stream `ofs`. It is assumed the endian-ness will be stored in a separate part of the file to help with decoding later. Returns `true` if all the data was successfully written.

OEBWriteDataLength

```
bool OEBWriteDataLength(OEPlatform::oeostream &, OEULongLong)
bool OEBWriteDataLength(OEPlatform::oeostream &, unsigned int)
```

OEBWriteOEBase

```
bool OEBWriteOEBase(OEPlatform::oeostream &, const OEBase &,
                    const std::string &,
                    const OEBinaryIOHandlerBase& hand)
```

OEBWritePOD

```
bool OEBWritePOD(OEPlatform::oeostream &, const OEBase &,
                 const std::string &)
```

OEBWriteTag

```
bool OEBWriteTag(OEPlatform::oeostream& ofs, const char *)
bool OEBWriteTag(OEPlatform::oeostream& ofs, const char *,
                 unsigned char tagtype)
```

OEBitIsPowerOf2

```
bool OEBitIsPowerOf2(unsigned int bitmask,
                     const bool allowZero = false)
```

Returns `true` if `bitmask` is a power of 2; otherwise, it returns `false`.

allowZero
: If this value is set to `true`, 0 is considered a power of 2. The default is `false`.

OEBitVectorAnd

```
OEBitVector OEBitVectorAnd(const OEBitVector& a, const OEBitVector& b)
```

Returns boolean conjunction (or 'and') of two *OEBitVectors*. The result will have the same number of bits as the larger of the two arguments.

OEBitVectorEqual

```
bool OEBitVectorEqual(const OEBitVector& a, const OEBitVector& b)
```

Determines whether two *OEBitVectors* are considered equal. The two *OEBitVectors* are only considered equal if they have the same size, *i.e.* `a.GetSize() == b.GetSize()`, and all of the bits have the same value.

OEBitVectorNot

```
OEBitVector OEBitVectorNot(const OEBitVector& bv)
```

Returns the boolean negation (or 'inversion') of an *OEBitVector*. The result will have the same number of bits as the argument.

OEBitVectorNumBytes

```
inline size_t OEBitVectorNumBytes(size_t bits)
```

Returns the number of words required to store the number of bits passed as the `bits` parameter, rounded up to a multiple of machine word size.

OEBitVectorNumWords

```
inline size_t OEBitVectorNumWords(size_t bits)
```

Returns the number of words required to store the number of bits passed as the `bits` parameter.

OEBitVectorOr

```
OEBitVector OEBitVectorOr(const OEBitVector& a, const OEBitVector& b)
```

Returns the boolean disjunction (or 'or') of two *OEBitVectors*. The result will have the same number of bits as the larger of the two arguments.

OEBitVectorSub

```
OEBitVector OEBitVectorSub(const OEBitVector& a, const OEBitVector& b)
```

Returns the boolean difference of two *OEBitVectors* This function returns an *OEBitVector* where each bit is set only if that position is set in the first operand and clear in the second. The result will have the same number of bits as the larger of the two arguments.

OEBitVectorXor

```
OEBitVector OEBitVectorXor(const OEBitVector& a, const OEBitVector& b)
```

Returns that Boolean exclusive or (or 'xor') of two *OEBitVectors*. This function returns an *OEBitVector* where each bit is set only if the equivalent bit positions in the two operands have different values. The result will have the same number of bits as the large of the two arguments.

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OECastData

```
template<class T>
const T& OECastData(OEBaseData* b)
```

OECheckHelp

```
bool OECheckHelp(const OEInterface& itf, int argc, char** argv,
                 bool help_for_no_args = true,
                 OEPlatform::oeostream& ostr = OEERR)
```

Return `true` if `--help` is specified on the commandline given by the `argc` and `argv` arguments. If `--help` was found, print out a help message to the stream specified by the `ostr` argument, by default, standard error. The `--help` flag can also be passed as an argument to cause detailed information to be printed about that parameter.

See also

Most users should not need to call this function directly. The *OEInterface* constructor and *OEParseCommandLine* call this function internally.

OEClearGridBoundary

```
bool OEClearGridBoundary(OESystem::OEScalarGrid& grid)
```

Assigns `0.0` to every grid point located at the edge of a 'grid' (*OEScalarGrid*).

OEColorCreate

```
OEColor OEColorCreate(double r, double g, double b)
OEColor OEColorCreate(double r, double g, double b, double a)
```

Creates a color with the specified color components that are mapped into the internal representation in the range `[0, 255]`.

r, g, b

The red, green and blue components of the color. These numbers have to be in the range `[0.0, 1.0]`. Setting a color to be opaque black is equivalent to *OEColorSet* (0.0, 0.0, 0.0).

a

The alpha value defines the transparency of a color and can be represented by a float value in the range `[0.0, 1.0]`. An alpha value of `1.0` means that the color is completely opaque and an alpha value of `0.0` means that the color is completely transparent. If the alpha value is not specified, then a completely opaque color is created.

Note

The components of the color are always clamped into the range `[0, 255]` that is the internal representation of the color.

See also

- *OEColor* class

OEColorGet

```
void OEColorGet(const OEColor &, float& r, float& g, float& b)
void OEColorGet(const OEColor &, double& r, double& g, double& b)
void OEColorGet(const OEColor &, float& r, float& g, float& b,
                float& a)
void OEColorGet(const OEColor &, double& r, double& g, double& b,
                double& a)
```

Converts the internal unsigned integer representation of the color components into floating point numbers in the range `[0.0, 1.0]`.

r, g, b, a
The red, green, blue and alpha components of the color.

See also

- *OEColor* class
- *OEColorSet* function

OEColorPacked

```
unsigned int OEColorPacked(unsigned int r, unsigned int g,
                           unsigned int b, unsigned int a = 255)
```

OEColorSet

```
void OEColorSet(OEColor &, double r, double g, double b)
void OEColorSet(OEColor &, double r, double g, double b, double a)
```

Sets a color with the specified color components that are mapped into the internal representation in the range `[0, 255]`.

r, g, b
The red, green and blue components of the color. These numbers have to be in the range `[0.0, 1.0]`. Setting a color to be opaque black is equivalent to *OEColorSet* (0.0, 0.0, 0.0).

a
The alpha value defines the transparency of a color and can be represented by a float value in the range `[0.0, 1.0]`. An alpha value of `1.0` means that the color is completely opaque and an alpha value of `0.0` means that the color is completely transparent. If the alpha value is not specified, then a completely opaque color is created.

Note

The components of the color are always clamped into the range `[0, 255]` that is the internal representation of the color.

See also

- *OEColor* class
- *OEColorGet* function

OEConfigure

```
bool OEConfigure(OEInterface& itf, OEPlatform::oeistream& istr)
bool OEConfigure(OEInterface& itf, const unsigned char* dataptr)
```

Provides a mechanism for adding parameters and child *OEInterface*'s to an *OEInterface* object, itf, by parsing a specifically formatted text file. The text file must be converted into a null terminated character array at compile time, and passed as the configdata to this function. The format of the config file is a series of parameter or category records.

When *OEConfigure* encounters a parameter record, it adds an *OEParameter* object to the *OEInterface*. A parameter record takes the following form.

```
!PARAMETER <name> [order priority]
  !TYPE <type>
  !ALIAS <alias>
  !BRIEF <brief description>
  !DEFAULT <default value>
  !REQUIRED <true or false>
  !VISIBILITY <visibility>
  !KEYLESS <keyless setting>
  !LEGAL_VALUE <value>
  !ILLEGAL_VALUE <value>
  !LEGAL_RANGE <hi value> <low value>
  !ILLEGAL_RANGE <hi_value> <low_value>
  !DETAIL
    <detailed description line 1>
    <detailed description line 2>
    <detailed description line 3>
    .
    .
    .
!END
```

The order of individual fields appear within the parameter record is unimportant. Each parameter record must begin with `!PARAMETER` and end with `!END`, and each record must have a `!TYPE` field. All other fields within the parameter record are optional. So the simplest possible parameter record is.

```
!PARAMETER <name> [order priority]
  !TYPE <type>
!END
```

When *OEConfigure* parses this record it does the equivalent of the following C++ code.

```
OEParameter* param = itf.AddParameter(<type>);
param->SetName(<name>);
param->SetOrderPriority([order priority]);
```

If [order priority] isn't specified, zero is assumed.

Legitimate values of <type> are

string
Creates a `std::string` parameter

double
Creates a double parameter

float
Creates a float parameter

bool
Creates a bool parameter

int
Creates an int parameter

file
Creates a *oeisstream* parameter

param_file
Creates a *oeisstream* parameter that is recognized as a text file holding parameter settings, by `OEParseCommandLine`.

The remaining fields in a parameter record are optional and do the following:

!ALIAS
alias

```
param->AddAlias(alias);
```

This field can appear multiple times.

!BRIEF
brief description

```
param->SetBrief(brief description);
```

This field can only appear once.

!DEFAULT
default value

```
param->SetStringDefault(default value);
```

This field can only appear once.

!REQUIRED
true or *false*

```
param->SetRequired(true or false);
```

This field can only appear once.

!VISIBILITY
visibility

Visibility must be either *simple*, *normal* or *hidden* This causes either:

```
param->SetVisibility(OEParamVisibility::Simple);
param->SetVisibility(OEParamVisibility::Normal);
param->SetVisibility(OEParamVisibility::Hidden);
```

to be called respectively. This field can only appear once in a parameter record.

!KEYLESS

keyless setting

```
param->SetKeyless(keyless setting);
```

The keyless setting must be a non-negative integer. This field can only appear once in a parameter record.

!LEGAL_VALUE

value

```
param->AddLegalValue(value);
```

Parameters of type bool cannot have a `!LEGAL_VALUE` field. This field can appear multiple times in a parameter record.

!ILLEGAL_VALUE

value

```
param->AddIllegalValue(value);
```

Parameters of type bool cannot have an `!ILLEGAL_VALUE` field. This field can appear multiple times in a parameter record.

!LEGAL_RANGE

high value low value

```
param->AddLegalRange(hi value, low value);
```

Parameters of type bool, string, file and file_param cannot have a `!LEGAL_RANGE` field. This field can appear multiple times in a parameter record.

!ILLEGAL_RANGE

high value low value

```
param->AddIllegalRange(hi value, low value);
```

Parameters of type bool, string, file and file_param cannot have an `!ILLEGAL_RANGE` field. This field can appear multiple times in a parameter record.

!DETAIL

multi-line

All the lines following the `!DETAIL` keyword up until a line beginning with another parameter record keyword are added to the parameter.

This field can only appear once.

!CATEGORY

Category record format

When *OEConfigure* encounters a category record it adds a child *OEInterface* object to the current *OEInterface*. A category record takes the following form:

```
!CATEGORY <name> [order priority]
  !DETAIL
    <detailed description line 1>
    [detailed description line 2]
```

(continues on next page)

(continued from previous page)

```
    [detailed description line 3]
    .
    .
    .
  !BRIEF <brief description>
  [Parameter Record]
  [Category Record]
!END
```

The order individual fields appear within the category record is unimportant. All fields within the category record are optional. So the following is the minimal category allowed:

```
!CATEGORY <name> [order priority]
!END
```

When parsed by *OEConfigure* is equivalent to:

```
OEInterface* child_itf = itf.AddInterface();
child_itf->SetName(<name>);
child_itf->SetOrderPriority([order priority]);
```

If [order priority] is not specified 0 is assumed.

Fields within the category record have the following meaning:

!DETAIL
multi-line

All the lines following the `!DETAIL` keyword up until a line beginning with another category record keyword are added to the parameter.

This field can appear once in a category record.

!BRIEF

brief description

```
param->SetBrief(brief description);
```

This field can only appear once.

parameter record
Parameter records can appear inside category records. Their format is the same as those outside the category record, however the parameter will be added to the child interface. A category record can hold any number of parameter records.

category record
Category records can be nested within category records, thus creating a child *OEInterface* of a child *OEInterface*. There is no limit to the depth of nesting, and any number of nested category records can appear within a category record.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OECopyArray

```
template<class T, class U>
void OECopyArray(T* dst, const U* src, size_t num)
template<class T>
void OECopyArray(T* dst, const T* src, size_t num, OETrueType)
template<class T, class U>
void OECopyArray(T* dst, const U* src, size_t num, OEBoolType)
```

OECopyBaseData

```
void OECopyBaseData(OEBase& dst, const OEBase& src)
```

Copy *generic data* from one *OEBase* to another.

dst
: The destination of the generic data. *OEBase.Clear* is first called on this object to remove all data prior to copying.

src
: The source of the generic data that is copied over.

OECopyBits

```
void OECopyBits(OEBitVector& bv, const std::vector<bool>& vb)
void OECopyBits(std::vector<bool>& vb, const OEBitVector& bv)
```

OECount

```
template<class T>
unsigned int OECount(OEIter<T>& iter)
template<class T>
unsigned int OECount(OEIterBase<T>* iter)
```

Count the number of elements in an iterator.

> **Note**
>
> These templated functions are only available in C++. Please see the *OECount* functions in **OEChem TK**.

These methods have the following behavior:

- The OEIter version resets the iterator to the first element.
- The OEIterBase version consumes the iterator.

OEDivideGrid

```
template<class GridType1, class GridType2>
bool OEDivideGrid(GridType1& grid1, const GridType2& grid2)
```

OEErrorLevelToString

```
std::string OEErrorLevelToString(unsigned int level)
```

Return a string representation of the constants in the *OEErrorLevel* namespace.

OEFileExtension

```
std::string OEFileExtension(const std::string& filename,
                            const std::string& ignore)
```

OEFileExtensions

```
std::vector<std::string> OEFileExtensions(const std::string& filename)
```

OEGetBitCounts

```
void OEGetBitCounts(const OEBitVector& a, const OEBitVector& b,
                    unsigned int* onlyA, unsigned int* onlyB,
                    unsigned int* bothAB)
void OEGetBitCounts(const OEBitVector& a, const OEBitVector& b,
                    unsigned int* onlyA, unsigned int* onlyB,
                    unsigned int* bothAB, unsigned int* neitherAB)
```

Count the number of bits when comparing the two *OEBitVector* arguments a and b. This function is optimized so that the bitvectors only need to be traversed once.

The counts are returned as the following arguments with the corresponding values.

```
*onlyA     =   a  & ~b;
*onlyB     =  ~a  &  b;
*bothAB    =   a  &  b;
*neitherAB = ~(a  &  b);
```

See also

These values correspond with the **a**, **b**, **c**, **d** values found on the Daylight theory page.

OEGetCenterAndExtents

```
void OEGetCenterAndExtents(const OESkewGrid& grid, float* center,
                           float* extents)
void OEGetCenterAndExtents(const OESystem::OEGridBase& grid,
                           float* center, float* extents)
```

OEGetColors

```
OEIterBase<OEColor>* OEGetColors(const std::vector<OEColor>& colors)
```

Returns the iterator from the vector of specified colors in the vector order, enabling setting a custom color iterator.

```
OEIterBase<OEColor>* OEGetColors(const OEColor& fromcolor,
                                 const OEColor& tocolor,
                                 unsigned int nrcolors)
```

Returns the iterator over *nrcolors* colors between the two given colors using a *OELinearColorGradient* class. See example in *Figure: The ten colors that are returned from OEYellow to OEDarkRed colors*.



Fig. 27: **The ten colors that are returned from OEYellow to OEDarkRed colors**

OEGetContourColor

```
OEColor OEGetContourColor(const OEGridAbstractBase& grid,
                          unsigned int idx)
```

Returns the color for the contour associated with 'idx' from a 'grid' (*OEGridAbstractBase*).

See also

- *OEAddContour*
- *OEGetContourLevel*
- *OENumContours*

OEGetContourLevel

```
float OEGetContourLevel(const OEGridAbstractBase& grid,
                        unsigned int idx)
```

Returns the level for the contour associated with 'idx' from a 'grid' (*OEGridAbstractBase*)

See also

- *OEAddContour*
- *OEGetContourColor*
- *OENumContours*

OEGetContrastColors

```
OEIterBase<OEColor>* OEGetContrastColors(bool whiteblack = false)
```

Returns the iterator over colors of maximum contrast depicted in *Figure: Colors of maximum contrast*.

whiteblack
: If true, then *OEWhite* and *OEBlack* are returned as the first and second color by the *OEGetContrastColors* function.



Fig. 28: **Colors of maximum contrast**



Fig. 29: **Colors of maximum contrast with white and black**

The 22 colors depicted in *Figure: Colors of maximum contrast with white and black* are called Kelly's colors. The order of colors in Kelly's list is designed to maximize contrast between colors in a set. The first nine colors (white, black, yellow, purple, orange, light blue, red, buff, grey) are selected to be discriminating even for people with color vision deficiency.

See also

- [Kelly-1965]
- [Green-Armytage-2010]

OEGetDataType

```
template<typename T>
const void* OEGetDataType()
template<typename T>
inline const void* OEGetDataType(T&)
template<>
const void* OEGetDataType<unsigned int>()
```

These template functions provide a convenient and efficient mechanism for run-time type identification (RTTI). A unique const void pointer is returned for each requested data type. Template classes which derive from a base class which has a data type identification interface can then be resolved by comparisons to data types returned by the *OEGetDataType* template function. Type comparisons are valid across compile units. Run-time type identification using this function is likely to be more efficient than built-in C++ RTTI.

OEGetDeepColors

```
OEIterBase<OEColor>* OEGetDeepColors()
```

Returns the iterator over eight deep colors depicted in *Figure: Deep colors*.



Deep colors

OEGetFileEndian

```
template<class T>
bool OEGetFileEndian(unsigned int)
template<>
bool OEGetFileEndian<int>(unsigned int byte)
template<>
bool OEGetFileEndian<unsigned int>(unsigned int byte)
template<>
bool OEGetFileEndian<float>(unsigned int byte)
template<>
bool OEGetFileEndian<double>(unsigned int byte)
template<>
bool OEGetFileEndian<short>(unsigned int byte)
template<>
bool OEGetFileEndian<oefpos_t>(unsigned int byte)
template<>
bool OEGetFileEndian<OEPlatform::OEHalfFloat>(unsigned int byte)
```

Given the endianness encoded in the `byte`, return whether this machine needs to swap bytes for the given data type `T`. Always returns `false` if `sizeof(T)` is 1. Will fail to compile if the size is greater than 1 and an explicit template specialization is not given.

See also

Designed to work in conjunction with *OEGetHostEndian*. The byte returned by *OEGetHostEndian* should be written to the output file, then *OEGetFileEndian* will determine whether the bytes of the given type `T` need to be swapped while being read by *OEBReadData*.

OEGetGridFileType

```
unsigned int OEGetGridFileType(const char* ext)
```

Returns a symbolic constant from the *OEGridFileType* namespace for the file extension 'ext'.

OEGetGridFormatExtension

```
const char* OEGetGridFormatExtension(unsigned int tag)
```

Returns a comma-separated list of possible file extensions corresponding to the specified parameter. The parameter, 'tag', should be drawn from the *OEGridFileType* namespace.

OEGetGridFormatString

```
const char* OEGetGridFormatString(unsigned int tag)
```

Returns the name of the file format associated with the symbolic constant 'tag' from the *OEGridFileType* namespace.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEGetHostEndian

```
template<class T>
unsigned char OEGetHostEndian()
template<>
unsigned char OEGetHostEndian<int>()
template<>
unsigned char OEGetHostEndian<unsigned int>()
template<>
unsigned char OEGetHostEndian<float>()
template<>
unsigned char OEGetHostEndian<double>()
template<>
unsigned char OEGetHostEndian<short>()
template<>
unsigned char OEGetHostEndian<oefpos_t>()
template<>
unsigned char OEGetHostEndian<OEPlatform::OEHalfFloat>()
```

Returns the endianness of this machine so it can be encoded to a binary output file. The byte is used to indicate whether multi-byte binary values will need to be swapped when read on a different machine.

See also

Designed to work in conjunction with *OEGetFileEndian*. The byte returned by *OEGetHostEndian* should be written to the output file, then *OEGetFileEndian* will determine whether the bytes of the given type T need to be swapped while being read by *OEBReadData*.

OEGetKappa

```
double OEGetKappa(double prefactor)
double OEGetKappa(const OEGaussianBase &)
```

OEGetLightColors

```
OEIterBase<OEColor>* OEGetLightColors()
```

Returns the iterator over eight light colors depicted in *Figure: Light colors*.



Light colors

OEGetMemPoolMode

```
unsigned int OEGetMemPoolMode()
```

OEGetOEBReturnString

```
const char* OEGetOEBReturnString(unsigned int idx)
```

OEGetPrefactor

```
double OEGetPrefactor(double kappa)
```

OEGetRadius

```
double OEGetRadius(const OEGaussianBase &)
double OEGetRadius(double prefactor, double width)
```

OEGetTag

Associates a text string tag with a corresponding unsigned integer tag. Because integer comparisons are often faster than character string comparisons, operations which require a potentially large number of data comparisons may benefit by using integer instead of string data. The *OEGetTag* functions serve to associate, store and retrieve string and integer tags.

```
const char* OEGetTag(unsigned int tag)
```

Retrieves a character string tag associated with an integer tag. If no string tag has been associated with the passed integer tag, a pointer to a null-terminated pointer character string is returned. The function should always return a non-zero pointer.

```
unsigned int OEGetTag(const std::string& tag)
```

Retrieves an integer tag associated with the character string stored in the STL string object. The first call to *OEGetTag* with a particular character string creates an association with an arbitrary unique integer, and returns the associated integer tag. Subsequent calls to *OEGetTag* with the same character string value return the associated integer tag. No guarantees are made regarding the allocation of integer values associated with character string values.

```
unsigned int OEGetTag(const char* tag)
```

Retrieves an integer tag associated with a constant character string. The first call to *OEGetTag* with a particular character string creates an association with an arbitrary unique integer, and returns the associated integer tag. Subsequent calls to *OEGetTag* with the same character string value return the associated integer tag. No guarantees are made regarding the allocation of integer values associated with character string values.

OEGetThreadSafe

```
bool OEGetThreadSafe()
```

Reports whether *OESystem* memory allocation uses thread-safe mutual exclusion to access its data structures. By default, this value is `true`. This allows *OEChem* molecules and other toolkit objects to be allocated and deallocated simultaneously in concurrent operating system threads. However, for applications that only have a single thread of execution, the use of thread-safe memory allocation can cause a small performance overhead on some platforms. Setting this value to `false`, using the *OESetThreadSafe* function can be used to instruct *OESystem* (and the toolkits built upon it) to use faster implementations when it is known to be safe.

OEGetVividColors

```
OEIterBase<OEColor>* OEGetVividColors()
```

Returns the iterator over eight vivid colors depicted in *Figure: Vivid colors*.



Vivid colors

OEGetWidth

```
double OEGetWidth(double kappa, double radius)
```

> **Attention**
>
> This function is available in C++ and Python only.

OEGridAvgVal

```
template<class GridType>
GridType OEGridAvgVal(const OEFixedGrid<GridType>& grid)
```

Returns the average value of the `grid`.

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEGridCopyContents

```
template<class GridType>
bool OEGridCopyContents(OEFixedGrid<GridType>& dest,
                        const OEFixedGrid<GridType>& src)
```

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEGridCopyGeometry

```
template<class GridObject1, class GridObject2>
bool OEGridCopyGeometry(GridObject1& dest, const GridObject2& src)
```

OEGridGetArch

```
const char* OEGridGetArch()
```

OEGridGetLicensee

```
bool OEGridGetLicensee(std::string& licensee)
```

OEGridGetPlatform

```
const char* OEGridGetPlatform()
```

OEGridGetRelease

```
const char* OEGridGetRelease()
```

Returns the release name of the *OEGrid* library being used. This returns a value similar to `1.0` for production versions of the library, and `1.0 debug` for the checking version of the library.

See also

- *OEGridGetVersion*.

OEGridGetSite

```
bool OEGridGetSite(std::string& site)
```

OEGridGetVersion

```
unsigned int OEGridGetVersion()
```

Returns the version number of the *OEGrid* library being used. This is an unsigned integer value indicating the date on which the library was built, for example `20020903`, for the 3rd of September 2002. This value should be used when reporting problems, and unlike the release string returned by the *OEGridGetRelease* function, may be used in comparisons if needed.

OEGridHasNonZeroValue

```
bool OEGridHasNonZeroValue(const OESystem::OESkewGrid& g)
bool OEGridHasNonZeroValue(const OESystem::OEScalarGrid& g)
```

Returns whether the data in the grid 'g' (*OESkewGrid* or *OEScalarGrid*) is zeroed out.

OEGridIsLicensed

```
bool OEGridIsLicensed(const char* feature = 0,
                      unsigned int* expdate = 0)
```

Determines whether a valid license file is present. This function may be called without a legitimate run-time license to determine whether it is safe to call any **OEGrid TK** functionality.

The 'feature' argument can be used to check for a valid license to **OEGrid TK** along with the following features: 'python', 'java', or 'clr' (for CSharp).

The 'expdate' argument can be used to obtain the expiration date of the current **OEGrid TK** license.

See also

- *OEChemIsLicensed* function

Attention

This function is available in C++ and Python only.

OEGridMaxVal

```
template<class GridType>
GridType OEGridMaxVal(const OEFixedGrid<GridType>& grid)
```

Returns the maximum value of the `grid`.

Attention

This function is available in C++ and Python only.

OEGridMinVal

```
template<class GridType>
GridType OEGridMinVal(const OEFixedGrid<GridType>& grid)
```

Returns the minimum value of the `grid`.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEGridSameGeometry

```
template<class GridObject1, class GridObject2>
bool OEGridSameGeometry(const GridObject1& grid1,
                        const GridObject2& grid2)
```

OEGridSizeMultiply

```
template<typename OutputType, typename X, typename Y, typename Z>
OutputType OEGridSizeMultiply(X xdim, Y ydim, Z zdim)
```

This template method is used by **OpenEye** toolkit grid handling to safely multiply the 3 dimensions of a grid together while ensuring it does not overflow the size of the `OutputType` integer type. If the multiplication of `xdim`, `ydim`, and `zdim` would result in an integer overflow, `0` will be returned instead. If the multiplication can be safely performed, the total resulting grid size is returned.

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEGridSetAllElements

```
template<class GridType>
void OEGridSetAllElements(OEFixedGrid<GridType>& grid,
                          const GridType& value)
```

OEHasTag

```
bool OEHasTag(unsigned int tag)
bool OEHasTag(const std::string& s)
```

OEInitAnnotationHandlers

```
void OEInitAnnotationHandlers(OESystem::OEBinaryIOHandlerBase& hand)
```

OEInitGridHandlers

```
void OEInitGridHandlers(OEChem::oemolstreambase& fs)
```

Adds the ability to read/write *OEScalarGrid* and *OESkewGrid* attached to molecules.

```
bool OEInitGridHandlers(OESystem::OEBinaryIOHandlerBase& b)
```

Adds the ability to read/write *OEScalarGrid* and *OESkewGrid* with any *OEBinaryIOHandlerBase*.

OEInitializeHandler

```
void OEInitializeHandler(OESystem::OEBinaryIOHandlerBase& hand)
```

OEInterpolateBetweenGrids

```
bool OEInterpolateBetweenGrids(OESystem::OEScalarGrid& dst,
                               const OESystem::OEScalarGrid& src)
```

Sets the values on 'dst' grid (*OEScalarGrid*) by linearly interpolating the values from 'src' grid (*OEScalarGrid*).

OEIsEmpty

```
template<class T>
unsigned int OEIsEmpty(OEIterBase<T>* iterbase)
```

Returns whether the iterator is empty and consumes the iterator.

```
bool has_hydrogens = !OEIsEmpty<OEAtomBase>(mol.GetAtoms(OEIsHydrogen()));
```

OEIsGrid

```
bool OEIsGrid(const OESystem::OEBase& b)
```

Returns whether 'b' is a *OEScalarGrid* or a *OESkewGrid*.

OEIsReadableGrid

```
bool OEIsReadableGrid(unsigned int type)
bool OEIsReadableGrid(const std::string& filename)
```

Returns `true` if the supplied file format is readable by *OEGrid*. The 'filename' can be a file name or a file extension. The 'type' parameter should be drawn from the *OEGridFileType* namespace.

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEIsSafeCast

```
template<class Ret, class Input>
inline bool OEIsSafeCast(Input arg, std::string& msg)
```

Returns `true` if the input value in `arg` can be safely cast to the output type `Ret` without loss of information. Returns `false` if there would be lost information, i.e., casting an negative number to an unsigned type. If `false` is returned, an error message is placed into the `msg` argument string.

OEIsSameTypeTester

```
OEFalseType OEIsSameTypeTester(...)
template<typename T>
OETrueType OEIsSameTypeTester(T &, T &)
```

OEIsWriteableGrid

```
bool OEIsWriteableGrid(unsigned int type)
bool OEIsWriteableGrid(const std::string& filename)
```

Returns `true` if the supplied file format is writable by *OEGrid*. The 'filename' can be a file name or a file extension. The 'type' parameter value should be drawn from the *OEGridFileType* namespace.

> **Attention**
>
> This function is available in C++ and Python only.

OELinearInterpolate

```
template<class GridType>
inline GridType OELinearInterpolate(const OEFixedGrid<GridType>& grid,
                                    float x, float y, float z,
                                    const GridType& outside_grid_val)
```

Returns the tri-linear interpolation of the value at `(x, y, z)` inside the `grid`. If the point falls outside the grid, *outside_grid_val* is returned instead.

OEMTZMapTypeFromString

```
unsigned int OEMTZMapTypeFromString(const std::string& maptype)
```

Converts the human readable string output by *OEMTZMapTypeToString* into the a MTZ map type. Returns *OEMTZMapType.None* upon failure.

See also

- *OEMTZMapType* namespace
- *OEReadMTZ* function
- *OEMTZMapTypeToString* function

See also

- *OEReadMTZ* function

OEMTZMapTypeToString

```
std::string OEMTZMapTypeToString (unsigned int maptype)
```

Converts a MTZ map type, such as *OEMTZMapType.Fdelwt* to a human readable string.

MapType	Human-Readable String
OEMTZMapType::Fo	“Fo”
OEMTZMapType::Fc	“Fc”
OEMTZMapType::TwoFoFc	“2Fo-Fc”
OEMTZMapType::FoFc	“Fo-Fc”
OEMTZMapType::ThreeFoTwoFc	“3Fo-2Fc”
OEMTZMapType::FiveFoThreeFc	“5Fo-3Fc”
OEMTZMapType::TwoFoFcSigmaA	“2Fo-Fc(SigmaA)”
OEMTZMapType::FoFcSigmaA	“Fo-Fc (SigmaA)
OEMTZMapType::FoSquared	“Fo*Fo”
OEMTZMapType::FoFom	“Fo*FOM”
OEMTZMapType::Fwt	“Fwt”
OEMTZMapType::Fdelwt	“Fdelwt”
OEMTZMapType::None	“None”

See also

- *OEMTZMapType* namespace
- *OEReadMTZ* function
- *OEMTZMapTypeFromString* function

OEMakeGaussians

```
bool OEMakeGaussians(OEGaussianContainerBase& gaussians,
                     const OEChem::OEMolBase& mol, double prefactor = 2.7)
```

OEMakeGridFromCenterAndExtents

```
template<class T>
bool OEMakeGridFromCenterAndExtents(OESystem::OEGrid<T>& fgrid,
                                    const float* center,
                                    const float* extents,
                                    float spacing)
bool OEMakeGridFromCenterAndExtents(OESystem::OEScalarGrid& fgrid,
                                    const float* center,
                                    const float* extents,
                                    float spacing)
```

Initiates the geometry of 'fgrid' from the midpoint ('center'), dimensions ('extents'), and length between grid points ('spacing').

This is useful for creating grids that are guaranteed to encapsulate an arbitrary set of coordinates. For example, the following code uses *OEGetCenterAndExtents* on the molecule and then uses *OEMakeGridFromCenterAndExtents* to construct the grid.

Initializing grid geometry from the coordinates of a molecule

```
float[] center = new float[3];
float[] extents = new float[3];
float spacing = 0.5f;
OEChem.OEGetCenterAndExtents(mol, center, extents);

OEScalarGrid grid = new OEScalarGrid();
OEGrid.OEMakeGridFromCenterAndExtents(grid, center, extents, spacing);
```

OEMakeHex

```
bool OEMakeHex(std::string& hex, const OEBitVector &,
               unsigned int columns = 0)
```

Generates a hexadecimal string representation of an *OEBitVector*.

```
bool OEMakeHex(std::string &, const std::vector<bool> &,
               unsigned int columns = 0)
```

Generates a hexadecimal string representation of a boolean vector.

See also

- *OEParseHex* function
- *OEBitVector.FromHexString* and *OEBitVector.ToHexString* method method

OEMakeMolecularGaussianGrid

```
bool OEMakeMolecularGaussianGrid(OESystem::OEScalarGrid& grid,
                                 const OEChem::OEMolBase& mol,
                                 float resolution)
```

Populates 'grid' (*OEScalarGrid*) with floating point values determined by Gaussians centered at the atoms in mol (*OEMolBase*). The grid will be resized to fit around the molecule with the specified `resolution`.

```
bool OEMakeMolecularGaussianGrid(OESystem::OEScalarGrid& grid,
                                 const OEChem::OEMolBase& mol)
```

Similar to the above function, however, the passed in grid dimension and location will be used. Care must be taken to ensure the molecule lies entirely within the grid. The `resolution` of the grid is determined by the passed in *OEGridBase.GetSpacing*.

OEMakeRegularGrid

```
bool OEMakeRegularGrid(OESystem::OEScalarGrid& gridout,
                       const OESystem::OESkewGrid& gridin,
                       float scale = 0.0f)
```

Regularizes the input grid, 'gridin' (*OESkewGrid*) into an orthogonal axis 'gridout' (*OEScalarGrid*). The full extents of the *OESkewGrid* will be transferred, potentially causing some data redundancy in the resulting *OEScalarGrid*.

OEMakeRegularSubGrid

```
bool OEMakeRegularSubGrid(OESystem::OEScalarGrid& gridout,
                          const OESystem::OESkewGrid& gridin,
                          const float* center, const float* extents,
                          float scale, bool reentrant)
bool OEMakeRegularSubGrid(OESystem::OEScalarGrid& gridout,
                          const OESystem::OEScalarGrid& gridin,
                          const float* center, const float* extents,
                          float scale, bool reentrant)
```

Regularize a portion of 'gridin' (*OESkewGrid*) into 'gridout' (that can be either *OEScalarGrid* or *OESkewGrid*)

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEMakeString

```
std::string OEMakeString(const char* format, ...)
```

Return a formatted string using `printf` formatting rules.

OEMaskGrid

```
template<class GridType1, class GridType2>
bool OEMaskGrid(GridType1& outgrid, const GridType2& mask_grid,
                unsigned int type = OEGridMaskType::Default)
```

OEMaskGridByCoordsAndRadii

```
bool OEMaskGridByCoordsAndRadii(OESystem::OEScalarGrid& grid,
                                const float* coords,
                                const float* radii,
                                unsigned int numcoords,
                                unsigned int type = OEGridMaskType::Default)
```

OEMaskGridByMolecule

```
bool OEMaskGridByMolecule(OESystem::OEScalarGrid& grid,
                          const OEChem::OEMolBase& mol,
                          unsigned int masktype, float contour)
bool OEMaskGridByMolecule(OESystem::OEScalarGrid& grid,
                          const OEChem::OEMolBase& mol,
                          unsigned int masktype = OEGridMaskType::Default)
```

OEMaskGridValues

```
bool OEMaskGridValues(float* vals, const float* cvals,
                      unsigned int count,
                      unsigned int type = OEGridMaskType::Default)
```

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEMoveArray

```
template<class T, class U>
void OEMoveArray(T* dst, const T* src, size_t num)
template<class T>
void OEMoveArray(T* dst, const T* src, size_t num, OETrueType)
template<class T, class U>
void OEMoveArray(T* dst, const U* src, size_t num, OEBoolType)
```

OEMultiplicationWillOverflow

```
template<typename NumType>
bool OEMultiplicationWillOverflow(NumType m, NumType n)
```

Returns `true` if the multiplication of `m` and `n` will overflow the integer number type `NumType`. Returning `false` indicates the numbers can be safely multiplied without incurring any undefined C++ behavior.

OEMultiplyGrid

```
template<class GridType1, class GridType2>
bool OEMultiplyGrid(GridType1& grid1, const GridType2& grid2)
```

OENumContours

```
unsigned int OENumContours(const OEGridAbstractBase& grid)
```

Returns the number of contours stored on 'grid' (*OEGridAbstractBase*).

See also

- *OEAddContour*
- *OEGetContourColor*
- *OEGetContourLevel*

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OENumberToString

```
std::string OENumberToString(const bool& x)
std::string OENumberToString(const std::string& x)
std::string OENumberToString(int value, int base = 10)
std::string OENumberToString(short value, int base = 10)
std::string OENumberToString(OELongLong number, int base = 10)
std::string OENumberToString(OEULongLong number, int base = 10)
std::string OENumberToString(unsigned int value, int base = 10)
std::string OENumberToString(unsigned short value, int base = 10)
std::string OENumberToString(float number, char f = 'f', int prec = 6)
std::string OENumberToString(double number, char f = 'f', int prec = 12)
std::string OENumberToString(long double number, char f = 'f',
                             int prec = 12)
bool OENumberToString(short value, std::string& numstr, int base = 10)
inline bool OENumberToString(int value, std::string& numstr,
                             int base = 10)
bool OENumberToString(OELongLong value, std::string& numstr,
                      int base = 10)
bool OENumberToString(OEULongLong value, std::string& numstr,
                      int base = 10)
bool OENumberToString(unsigned int value, std::string& numstr,
                      int base = 10)
bool OENumberToString(unsigned short value, std::string& numstr,
                      int base = 10)
bool OENumberToString(double value, std::string& numstr,
                      char format = 'f', int precision = 6)
bool OENumberToString(long double value, std::string& numstr,
                      char format = 'f', int precision = 6)
```

Returns a string representation of the number `value` passed in. For integral types, the base should be between 2 and 36. For floating point types, the number of decimal places can be chosen with the `prec` argument. The format parameter, `f` should be one of the following: `f`, `g`, or `e`.

The overload that returns a boolean value can be used to detect whether the function fails. The overloads that return the string will return an empty string if the function fails.

OEParseCommandLine

```
bool OEParseCommandLine(OEInterface& itf, int argc, char** argv,
                        const unsigned char* requirements = NULL,
                        unsigned int error_level = OEErrorLevel::Error,
                        bool checkhelp = true)
bool OEParseCommandLine(OEInterface& itf, const char* version,
                        int argc, char** argv,
                        const unsigned char* requirements = NULL,
                        unsigned int error_level = OEErrorLevel::Error,
```

(continues on next page)

(continued from previous page)

```
                        bool checkhelp = true)
```

Sets parameters in the 'itf' object by parsing the command line, 'argc' and 'argv'. This function returns `true` if it was successful and `false` otherwise. Errors are reported at 'error_level' to *OEThrow*.

The `version` argument specifies what should be printed whenever the `--version` is found in `argv`.

By default, `checkhelp` is true, forcing *OECheckHelp* to be called at the very beginning of this function. If `checkhelp` is true and the `--help` parameter is found, the `exit` system call is called to exit the process with a `0` exit code.

Parameters are entered in key-value pairs after the executable name on the command line.

```
program <parameter1 key> <parameter1 value> <parameter2 key>  <parameter2 value>
```

The order in which the parameter key-value pairs appear on the command line is unimportant, except when the same key is specified twice in which case the second value specified is used.

```
program <parameter2 key> <parameter2 value> <parameter1 key> <parameter1 value>
```

is the same as the preceding example.

The following special case rules also apply when parsing the command line.

1. Boolean parameter keys can optionally not be followed by a boolean value in which case the boolean parameter is set to true

   ```
   program -b true -x 5
   ```

   and

   ```
   program -b -x 5
   ```

   are equivalent, provided that '-b' is a boolean parameter.

2. Parameters of type 'param_file' will be assumed to be text files containing key-value parameter pairs for the *OEInterface* object, 'itf'. These files will be parsed and the parameters set accordingly. The format of these files should be one key-value pair per line. Blank lines and lines beginning with # will be ignored. If a parameter is set both in a parameter file and on the command line, the command line setting will be used.

3. Parameters with non-zero keyless values (see *OEParameter.GetKeyless* method) can be entered at the end of the command line as a value without a key. The format is as follows

   ```
   program [<key> <value>]... <keyless1 value> [<keyless2 value>]...
   ```

Any number of key-value pairs can be entered on the command line before the keyless values. Parsing from left to right the first argument on the command line not recognized as a key-value pair is assigned to the parameter with the keyless value 1. Additional arguments beyond the first are assigned to the parameter with keyless value 2, 3, 4 and so on. If the itf object does not have a parameter with the appropriate keyless value or has more than one parameter with the appropriate keyless value an error will be reported and the function will return false.

Note

There is a potential ambiguity in the command line when the last key-value pair specified before the first keyless parameter is a boolean, as in the following example.

```
program -b true keylessvalue
```

If '-b' is a boolean parameter true could either be the setting of '-b' or the first keyless parameter. *OEParseCommandLine* resolves this by always assuming that true, or any valid boolean setting (*i.e.*, `true/yes/on/t/y` or `false/no/off/f/n`), is a setting for the boolean parameter '-b' and NOT the first keyless value.

After *OEParseCommandLine* parses the command line it then checks that all required parameters have been set. If all required parameters have not been set an error is issued and the function returns false.

The following are the types of requirements:

single parameter requirements

This type of requirement forces that a particular parameter must be specified on the command line, in a parameter file, or by a default value. This requirement is turned on using the SetRequired method of the *OEParameter* class (or the `!REQUIRED` field on *OEConfigure*).

multi-parameter requirements

This type of requirement can include inter-dependent settings of parameters. These requirements are specified by passing requirements to this function, which is a specially formatted text file that has been converted into a null terminated character array. The format is a series of one or more requirement records of the following format.

```
!REQUIREMENT
    !OPTION [[!]key [value] [[!]key [value]...
    !OPTION [[!]key [value] [[!]key [value]...
    .
    .
    .
    !ERROR_MSG
      <error message line 1>
      <error message line 2>
      .
      .
      .
!END
```

One or more `!OPTION` fields may appear in a requirement record. Each `!OPTION` field is a list of keys, optionally with values, to require. The rules for the format of the `!OPTION` line are as follows.

1. If a key appears on the `!OPTION` line, but is not followed by a parameter value, then that parameter must have a valid setting (*e.g.*, the parameter method *OEParameter.IsSet* must return `true`).
2. If a key appears on the `!OPTION` line followed by a value, then that parameter must be set to the specified value.
3. If a key appears on the `!OPTION` line preceded by a `!`, but is not followed by a parameter value, then that parameter must not have a valid setting (*e.g.*, the parameter method *OEParameter.IsSet* must return `false`).

If any `!OPTION` field of a requirement record is satisfied, the entire requirement is satisfied. No !OPTION in the requirement record is satisfied, the error message is sent to *OEThrow* and the function returns false.

The following is an example parameter record:

```
!REQUIREMENT
    !OPTION -a 5
    !OPTION !-b
    !OPTION -a -b !-c 1
    !ERROR_MSG
      The command line must satisfy one of the following criterion
      1) -a is set to 5
      2) -b is not set
```

(continues on next page)

(continued from previous page)

```
      3) -a and -b are set and -c is not set to 1
!END
```

This requirement record requires that the settings in itf follow the rules listed between `!ERROR_MSG` and `!END`. If it does not those lines are sent to *OEThrow* and the function returns `false`.

OEParseCommandLineLW

```
bool OEParseCommandLineLW(OEInterface& itf, int argc, char** argv,
                          const unsigned char* requirements = NULL,
                          unsigned int error_level = OEErrorLevel::Error)
bool OEParseCommandLineLW(OEInterface& itf, const char* version,
                          int argc, char** argv,
                          const unsigned char* requirements = NULL,
                          unsigned int error_level = OEErrorLevel::Error)
```

Set parameters in the 'itf' object by parsing the command line, 'argc' and 'argv'. This is the same purpose as *OEParseCommandLine*, however, this function has different rules for parsing the command line, most notably nothing is entered as a key-value pair, and is in general designed for small lightweight interfaces, hence the *LW* in the function name.

The `version` argument specifies what should be printed whenever the `--version` is found in `argv`.

When using this function all parameters of itf must fall into one of two categories.

- A boolean parameter with a default value of false and a name beginning with a dash followed by a single character.
- A keyless parameter (*i.e.* a parameter with a non-zero keyless value).

and the command line should be of the following form.

```
program [boolean parameter keys] [keyless parameter values]
```

boolean parameter keys

Boolean parameters with keys listed listed in this part of the command line will be set to true (those not appearing will use their default value which must be `false`). The characters of the boolean keys may also be combined into a single argument on the command line. This is easiest to illustrate in the following example:

```
program -a -b -c [keyless parameter values]
```

and

```
program -abc [keyless parameter values]
```

and

```
program -c -ba [keyless parameter values]
```

Are all equivalent, provided '-a', '-b' and '-c' are boolean parameters with default values of `false`.

keyless parameter values

The first argument not recognized as a boolean parameter will be assigned as the value of the parameter with a keyless value of 1, the second to the parameter with keyless value of 2, etc. If no parameter have the appropriate keyless value, or more than one have the required keyless value are error is reported, and the function returns `false`.

This function returns true if it was successful and `false` otherwise. Errors are reported at 'error_level' to *OEThrow*.

OEParseHex

```
bool OEParseHex(std::vector<bool>& , const std::string&)
```

Converts a hexadecimal string to boolean vector.

```
bool OEParseHex(OEBitVector& , const std::string&)
```

Converts a hexadecimal string to a *OEBitVector* object.

See also

- *OEMakeHex* function
- *OEBitVector.FromHexString* and *OEBitVector.ToHexString* method method

OEParseParameterFile

```
bool OEParseParameterFile(OEInterface& itf,
                          OEPlatform::oeistream& istr,
                          bool over_ride = true,
                          unsigned int error_level = OEErrorLevel::Warning)
```

OEReadCIFGridFile

```
bool OEReadCIFGridFile(const std::string& filename,
                       OESystem::OESkewGrid& sgrid,
                       unsigned int maptype)
```

Read a structure factor CIF file into an OpenEye skew grid. The grid requires that the proper column name be present to extract the appropriate reflection data.

The following map types can be read using predefined operations on the sfCIF data:

MapType	Description
OEMTZMapType.Fo	Observed Reflections
OEMTZMapType.Fc	Map generated from calculated model
OEMTZMapType.TwoFoFc	2Fo - Fc
OEMTZMapType.ThreeFoTwoFc	3Fo - Fc
OEMTZMapType.FiveFoThreeFc	5Fo - 3Fc
OEMTZMapType.TwoFoFcSigmaA	2Fo - Fc Sigma-A weighted
OEMTZMapType.FoFcSigmaA	Fo - Fc Sigma-A weighted
OEMTZMapType.FoSquared	Fo*Fo
OEMTZMapType.FoFom	Fo * figure of merit
OEMTZMapType.Fwt	2mFo - DFc (normally)
OEMTZMapType.Fdelwt	mFo - DFc (normally)
OEMTZMapType.None	No reflections (used as a status flag)

OEReadGrid

```
bool OEReadGrid(const std::string& filename,
                OESystem::OESkewGrid& grid)
bool OEReadGrid(const std::string& filename,
                OESystem::OEScalarGrid& grid)
bool OEReadGrid(OEPlatform::oeistream& ifs,
                OESystem::OESkewGrid& grid, unsigned int filetype)
bool OEReadGrid(OEPlatform::oeistream& ifs,
                OESystem::OEScalarGrid& grid, unsigned int filetype)
```

Reads the contents of the file into grid (*OESkewGrid*, *OEScalarGrid*).

OEReadMTZ

```
bool OEReadMTZ(const std::string& filename,
               OESystem::OESkewGrid& sgrid, unsigned int maptype)
```

Read an MTZ (crystallographic reflection format) into an OpenEye skew grid. MTZ grids require that the proper column names be known to extract the appropriate reflection data. This form of *OEReadMTZ* automatically chooses appropriate columns from the MTZ file to generate the requested map type. *OEReadMTZ* understands common refinement packages including REFMAC5, BUSTER and PHENIX. If the columns cannot be found or if the map cannot be generated *OEReadMTZ* returns false.

The following map types can be generated using predefined operations on the MTZ column data:

MapType	Description
OEMTZMapType.Fo	Observed Reflections
OEMTZMapType.Fc	Map generated from calculated model
OEMTZMapType.TwoFoFc	2Fo - Fc
OEMTZMapType.ThreeFoTwoFc	3Fo - Fc
OEMTZMapType.FiveFoThreeFc	5Fo - 3Fc
OEMTZMapType.TwoFoFcSigmaA	2Fo - Fc Sigma-A weighted
OEMTZMapType.FoFcSigmaA	Fo - Fc Sigma-A weighted
OEMTZMapType.FoSquared	Fo*Fo
OEMTZMapType.FoFom	Fo * figure of merit
OEMTZMapType.Fwt	2mFo - DFc (normally)
OEMTZMapType.Fdelwt	mFo - DFc (normally)
OEMTZMapType.None	No reflections (used as a status flag)

OEReadOldStyleGrid

```
bool OEReadOldStyleGrid(OEPlatform::oeistream& istr,
                        OEMultiGrid<float>& grid)
```

OERegisterMolParameters

```
bool OERegisterMolParameters()
```

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OERegisterOEBInitializer

```
void OERegisterOEBInitializer(OEBHandInitializer initializer)
```

OERotate

```
void OERotate(OESystem::OESkewGrid& g, const float* m)
```

```
void OERotate(OESystem::OESkewGrid& g, const double* m)
```

In functions above `m` is a 9-element array in row-major order

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OESNPrintf

```
int OESNPrintf(char* buffer, size_t bufferSize, const char* format, ...)
```

Cross platform implementation of the safe `snprintf` C function. Provided as a performance improvement over *OEMakeString*. *OEMakeString* should still be preferred as it will automatically handle memory management.

Warning

Considered an advanced performance tuning API that may change in the future.

OESetMemPoolMode

```
void OESetMemPoolMode(unsigned int mode)
```

OESetThreadSafe

```
void OESetThreadSafe(bool)
```

Controls whether the internal memory allocation and deallocation routines used by *OEChem* (and other OpenEye toolkits) should use slower thread-safe implementations, or faster single-threaded versions. By default, this value is `true`. This allows *OEChem* molecules and other toolkit objects to be allocated and deallocated simultaneously in concurrent operating system threads. However, for applications that only have a single thread of execution, the use of thread-safe mutual exclusion can cause a small performance overhead on some platforms. The current setting can be inspected by calling the *OEGetThreadSafe* function.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OESetupGrid

```
template <class GridObject> bool OESetupGrid(GridObject& grid,
                                             float* xyz, unsigned int N,
                                             float buffer = 0)
```

OESigmaNormalizeGrid

```
float OESigmaNormalizeGrid(OESystem::OESkewGrid& g)
float OESigmaNormalizeGrid(OESystem::OEScalarGrid& g)
```

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStaticGetTag

```
template<OEGetTagName GetTagName>
unsigned int OEStaticGetTag()
```

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringCSVJoin

```
std::string OEStringCSVJoin(const std::vector<std::string>& fields,
                            char delim = ',', char quote = '"')
```

Returns a string containing all the values of `fields` in the comma-separated-value, CSV, format specified by RFC 4180. The fields will only be quoted and escaped if necessary to avoid excess file size. The delimiter and quoting character can be changed to alternative characters, e.g., a tab delimiter for tab separated files.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringCSVQuote

```
std::string OEStringCSVQuote(const std::string& s, char delim = ',',
                             char quote = '"')
```

Returns the string `s` optionally surrounded by quotes, `quote`, only if the `delim` character occurs in the string or the first character of the string is the `quote` character. If quoted, instances of the `quote` character in the string will be escaped by inserting inserting another `quote` next to it. This is the quoting rules as defined by RFC 4180 for the comma-separated-value, CSV, format. This is a low-level CSV function, most users can just use *OEStringCSVJoin* and *OEStringCSVTokenize*.

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringCSVTokenize

```
bool OEStringCSVTokenize(std::vector<std::string>& tokens,
                         const std::string& text, char delim = ',',
                         char quote = '"')
```

Parse the `text` string into a number of `tokens` delimited according to the rules of the comma-separated-value, CSV, format specified by RFC 4180. The delimiter and quoting character can be changed to alternative characters, e.g., a tab delimiter for tab separated files.

> **Attention**
>
> This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringCollapseQuotes

```
std::string OEStringCollapseQuotes(const std::string& str,
                                   const std::string& quotes = "\"")
```

Replaces all the instance of two consecutive `quotes` in `str` with a single instance of that quote. The default is a double-quote, the standard for comma-separated-value, CSV, files. This is a low-level CSV function, most users can just use *OEStringCSVJoin* and *OEStringCSVTokenize*.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringEquals

```
bool OEStringEquals(const std::string& a, const std::string& b)
```

Returns whether the two given string is equivalent using case-insensitive comparison.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringJoin

```
std::string OEStringJoin(const std::list<std::string>& strgrp,
                         const std::string & delim = " ",
                         bool trailing_delim = true)
std::string OEStringJoin(const std::vector<std::string>& strgrp,
                         const std::string & delim = " ",
                         bool trailing_delim = true)
```

This function returns a new string created by concatenating the contents of the specified group of strings `strgrp` together delimited by the specified parameter `delim`. If `trailing_delim` is `true`, one additional delimiter will be appended to the string after the last member of `strgrp`.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringLower

```
std::string OEStringLower(const std::string &)
```

Returns a copy of the specified string with all of its letters in lower-case format.

See also

- *OEStringUpper* function

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringMatch

```
bool OEStringMatch(const std::string& str, const std::string& pat)
```

Returns whether or not the specified pattern string ('pat') matches the specified search string ('str'). String matching is case-sensitive and wildcards are limited to no more than two per pattern.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringSimplifyWhiteSpace

```
std::string OEStringSimplifyWhiteSpace(const std::string &)
```

Returns a copy of the specified string parameter which has had all contiguous whitespace characters converted to a single `' '`.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringStripQuotes

```
std::string OEStringStripQuotes(const std::string& s,
                                const std::string& quotes = "\"")
```

Removes the leading and trailing character from the string `s` only if the character is the same and is in the parameter `quotes`. This is a low-level CSV function, most users can just use *OEStringCSVJoin* and *OEStringCSVTokenize*.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringStripWhiteSpace

```
std::string OEStringStripWhiteSpace(const std::string &)
```

Returns a copy of the specified string parameter which has had all whitespace characters at the beginning and the end of the text string removed.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringTabsToSpaces

```
std::string OEStringTabsToSpaces(const std::string& s)
```

Return the string `s` with all the tab characters expanded to 4 space characters.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringToNumber

```
bool OEStringToNumber(const std::string& str, bool& x)
bool OEStringToNumber(const std::string& str, std::string& x)
bool OEStringToNumber(const std::string& numstr, float& value)
bool OEStringToNumber(const std::string& numstr, double& value)
bool OEStringToNumber(const std::string& numstr, int& value,
                      int base = 10)
bool OEStringToNumber(const std::string& numstr, short& value,
                      int base = 10)
bool OEStringToNumber(const std::string& numstr, OELongLong& value,
                      int base = 10)
bool OEStringToNumber(const std::string& numstr, OEULongLong& value,
                      int base = 10)
bool OEStringToNumber(const std::string& numstr, unsigned int& value,
                      int base = 10)
bool OEStringToNumber(const std::string& numstr, unsigned short& value,
                      int base = 10)
```

Attempts to convert the specified string into a numerical value in the specified base ('base') and store the generated value in the format appropriate to the passed parameter ('value'). The function returns whether or not this conversion was successful. If the function fails, 'value' is set to `0`.

See also

- Example program *SDFilter.cs*

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringTokenize

```
bool OEStringTokenize(std::list<std::string>& strgrp,
                      const std::string& str,
                      const std::string & delim = " \t\n",
                      bool concat_delim = true)
bool OEStringTokenize(std::vector<std::string>& strgrp,
                      const std::string& str,
                      const std::string & delim = " \t\n",
                      bool concat_delim = true)
```

Break ups the specified input string ('str') into individual tokens based on the specified delimiters ('delim'). The resulting tokens are stored in the specified string group (strgrp). This function always returns `true`. If *concat_delim* is `true`, the default, consecutive delimiters will be treated as a single delimiter. For example, for parsing CSV files *concat_delim* should be set to `false` to allow for empty fields.

See also

- Example program *CSV2SDF.cs*

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringTokenizeQuoted

```
bool OEStringTokenizeQuoted(std::list<std::string>& strgrp,
                            const std::string& str,
                            const std::string & delim = " \t\n",
                            const std::string & quote = "\'\"",
                            bool concat_delim = true)
bool OEStringTokenizeQuoted(std::vector<std::string>& strgrp,
                            const std::string& str,
                            const std::string & delim = " \t\n",
                            const std::string & quote = "\'\"",
                            bool concat_delim = true)
```

Break up the specified input string `str` into individual tokens based on the specified delimiters `delim` while maintaining the integrity of quoted text specified by quote delimiters `quote`. The resulting tokens are stored in the specified string group `strgrp`. This function always returns `true`.

The `concat_delim` parameter specifies whether consecutive delimiters should be treated as a single delimiter. If `false`, empty strings will be output into `strgrp` between consecutive delimiters.

The defaults of this function are set up for parsing whitespace separated fields usually found in configuration files. For parsing comma separated or tab separated file formats, the *OEStringCSVTokenize* will be easier to use.

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific

functionality instead.

OEStringTranslate

```
std::string OEStringTranslate(const std::string &)
```

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OEStringUpper

```
std::string OEStringUpper(const std::string &)
```

Returns a copy of the specified string with all of its letters in upper-case format.

See also

- *OEStringLower* function

OESubtractGrid

```
template<class GridType1, class GridType2>
bool OESubtractGrid(GridType1& grid1, const GridType2& grid2)
```

OETanimoto

```
double OETanimoto(const OEBitVector& a, const OEBitVector& b)
```

Calculates the `Tanimoto similarity coefficient` between two *OEBitVectors*. This is defined as the number of bits that the two bitmaps/fingerprints have in common divided by the total of bits set in either. This function does not require that the two *OEBitVectors* be the same size, the bits lacking from the shorter *OEBitVector* being interpreted as all zeros. This function always returns `1.0`, if the two *OEBitVectors* are equal, even if they are both empty.

OETranslate

```
void OETranslate(OESystem::OESkewGrid& g, const float* t)
void OETranslate(OESystem::OESkewGrid& g, const double* t)
```

Attention

This functionality is only available in the C++ toolkits. For other languages, we recommend using language-specific functionality instead.

OETrimToSigFigs

```
std::string OETrimToSigFigs(const std::string& numstr)
```

Strips trailing zeros from a string representing a decimal number. Nothing is stripped if no decimal point is given. A single zero character is left if only zeros follow the decimal point. The function is smart enough to properly deal with scientific notation formats. The following table is an example of inputs and outputs to the function.

Input	Output
1.00000	1.0
1.0	1.0
3.141600	3.1416
3.141600E+00	3.1416E+00

OEUnknownOEBDataTagName

```
const char* OEUnknownOEBDataTagName()
```

OEWriteGrid

```
bool OEWriteGrid(const std::string& filename,
                 const OESystem::OESkewGrid& grid)
bool OEWriteGrid(const std::string& filename,
                 const OESystem::OEScalarGrid& grid)
bool OEWriteGrid(OEPlatform::oeostream& ofs,
                 const OESystem::OESkewGrid& grid,
                 unsigned int filetype)
bool OEWriteGrid(OEPlatform::oeostream& ofs,
                 const OESystem::OEScalarGrid& grid,
                 unsigned int filetype)
```

Writes a grid (*OESkewGrid*, *OEScalarGrid*) to a file.

OEWriteHtml

```
bool OEWriteHtml(const OEInterface& itf, OEPlatform::oeostream& ostr,
                 std::string program_name, std::string program_version)
```

OEWriteInterfaceConfigurationFile

```
bool OEWriteInterfaceConfigurationFile(const OEInterface& itf,
                                       OEPlatform::oeostream& ostr)
```

OEWriteOldStyleGrid

```
bool OEWriteOldStyleGrid(OEPlatform::oeostream& ostr,
                         const OEFixedGrid<float>& grid)
bool OEWriteOldStyleGrid(OEPlatform::oeostream& ostr,
                         const OEFixedMultiGrid<float>& grid,
                         unsigned int cur_grid = 0)
```

OEWriteSettings

```
bool OEWriteSettings(const OEInterface& itf,
                     OEPlatform::oeostream& ostr = OEOUT,
                     bool include_default = true,
                     std::string mode = "")
```

The function writes the current setting of 'itf' to 'ostr'. If 'include_default' is `true` all parameter will be listed, otherwise only parameters specified by the user will be listed.

The format of this output is such that if 'ostr' is a file stream, the created file can be used as a parameter file (*i.e.* passed to a param_file parameter) in a subsequent run. The function returns `true` if it succeeds and `false` otherwise.

'mode' specifies a name of an optimized parameter set for OpenEye's applications.

OEBitVector Operators

4.4 Preliminary OESystem API

4.4.1 Preliminary OESystem Classes

OEBoolParameter

```
class OEBoolParameter : public OEPODParameter<bool>
```

The *OEBoolParameter* represents parameter that has `boolean` value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*

- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEBoolParameter()
OEBoolParameter(const std::string& name)
OEBoolParameter(const std::string& name, const bool default)
OEBoolParameter(const OEBoolParameter&)
```

Default and copy constructors.

Constructs an *OEBoolParameter* instance using the specified set of parameters.

OEConstParameter

```
class OEConstParameter : public OEUIntParameter
```

The *OEConstParameter* represents parameter that has `unsigned` value. Unlike *OEUIntParameter*, however, this parameter requires a set of constants defined in terms of <unsigned, std::string> pairs as legal entries.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*

- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

The *OEConstParameter* class defines the following public methods:

- *AddLegalEntry*

Constructors

```
OEConstParameter()
OEConstParameter(const std::string& name)
OEConstParameter(const std::string& name, const unsigned defValue,
                 const std::string& defKey)
OEConstParameter(const OEConstParameter&)
```

Default and copy constructors.

Constructs an *OEConstParameter* instance using the specified set of parameters.

Assignment operator.

AddLegalEntry

```
bool AddLegalEntry(const unsigned value, const std::string& key)
```

Adds a legal pair, defining valid constants for the parameter.

OEDoubleParameter

```
class OEDoubleParameter : public OEPODParameter<double>
```

The *OEDoubleParameter* represents parameter that has `double` value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*

- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEDoubleParameter()
OEDoubleParameter(const std::string& name)
OEDoubleParameter(const std::string& name, const double default)
OEDoubleParameter(const OEDoubleParameter&)
```

Default and copy constructors.

Constructs an *OEDoubleParameter* instance using the specified set of parameters.

OEFloatParameter

```
class OEFloatParameter : public OEPODParameter<float>
```

The *OEFloatParameter* represents parameter that has `float` value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEFloatParameter()
OEFloatParameter(const std::string& name)
OEFloatParameter(const std::string& name, const float default)
OEFloatParameter(const OEFloatParameter&)
```

Default and copy constructors.

Constructs an *OEFloatParameter* instance using the specified set of parameters.

OEIntParameter

```
class OEIntParameter : public OEPODParameter<int>
```

The *OEIntParameter* represents parameter that has `int` value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

Constructors

```
OEIntParameter()
OEIntParameter(const std::string& name)
OEIntParameter(const std::string& name, const int default)
OEIntParameter(const OEIntParameter&)
```

Default and copy constructors.

Constructs an *OEIntParameter* instance using the specified set of parameters.

OEMultiParameter

```
template<class TT>
class OEMultiParameter : public OESystem::OEPointerParameter<TT>
```

OWMultiParameter represents parametera that have three different representations of the values. The three representations are an unsigned , a std::string , and an object representation.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*

- *IsSet* and *IsSetToString*

The *OEMultiParameter* class defines the following public methods:

- *AddLegalEntry*
- *GetDefault* and *SetDefault*
- *GetSetting*
- *GetValue* and *SetValue*

Constructors

```
OEMultiParameter()
OEMultiParameter(const std::string& name, const bool customAllowed)
OEMultiParameter(const OEMultiParameter&)
```

Default and copy constructors.

Constructs an OEMultiParameter instance using the specified set of parameters. The second argument in the second constructor represents if a custom object (not already defined as a legal entry) could be used as a parameter.

Assignment operator.

AddLegalEntry

```
bool AddLegalEntry(const unsigned idxKey, const std::string& strKey,
                   const TT& lib)
```

Add new legal entry for this parameter. The three argument value represents the three representations of the same legal value.

GetDefault

```
TT* GetDefault() const
```

Returns the current default value of the parameter.

GetSetting

```
TT* GetSetting() const
```

Returns the current setting of the parameter.

GetValue

```
TT* GetValue() const
```

Returns the current user-defined value of the parameter.

SetDefault

```
bool SetValue(const unsigned)
```

Sets the default value of the parameter.

SetValue

```
bool SetValue(const unsigned)
bool SetValue(const std::string&)
bool SetValue(const TT&)
bool SetValue(OEPlatform::oeistream& ifs)
```

Sets the user-defined value of the parameter.

OEOptions

```
class OEOptions
```

The *OEOptions* is an abstract base class. The Options represent a collection of *OEParameter* class that defines the set of parameters associated with settings of various objects or calculations.

See also

- *OESimpleAppOptions* class

Code Examples

- *Define parameters* example

The *OEOptions* class defines the following public methods:

- *AddDetail* and *GetDetail*
- *AddOption*
- *AddParameter*
- *AddParameterPrefix*
- *AddParameterSuffix*
- *Clear*
- *DeleteOptions*
- *DeleteParameter*
- *FindDupParameters*
- *GetBrief* and *SetBrief*
- *GetName* and *SetName*
- *GetOption*
- *GetOptions*
- *GetOrderPriority* and *SetOrderPriority*
- *GetParameter*
- *GetParameters*
- *HasDupParameters*
- *HasOption*

- *HasParameter*
- *SetParameterVisibility*
- *UpdateValues*

AddDetail

```
bool AddDetail(const std::string& detail)
```

Adds a line to the detailed description of the options.

AddOption

```
OEOptions* AddOption(const OEOptions& opts)
```

Adds a copy of the specified options as sub-options to this options.

AddParameter

```
OEParameter* AddParameter(const OEParameter&)
```

Adds a copy of the specified parameter to the options.

AddParameterPrefix

```
bool AddParameterPrefix(const std::string& prefix,
                        const bool underScore = true)
```

Add the specified prefix to the name of all the parameters.

AddParameterSuffix

```
bool AddParameterSuffix(const std::string& suffix)
```

Add the specified suffix to the name of all the parameters.

Clear

```
void Clear()
```

Removes all parameters and sub-options.

DeleteOptions

```
bool DeleteOptions(OEOptions&)
```

Deletes the specified sub-options, if it belongs to this options.

DeleteParameter

```
bool DeleteParameter(OEParameter& param)
```

Deletes the specified parameter, if it belongs to this options.

FindDupParameters

```
OEIterBase<OEParameter>* FindDupParameters(const bool visibleOnly = true)
OEIterBase<const OEParameter>* FindDupParameters(const bool visibleOnly = true) const
```

Returns duplicate parameters. Two parameters are considered duplicates if they have a common name. If `visibleOnly` is set to `True`, parameters with visibility *OEParamVisibility.Hidden* are ignored from this list. This operation is always recursive to the associated sub-options.

GetBrief

```
std::string GetBrief() const
```

See *SetBrief* method.

GetDetail

```
OEIterBase<const std::string>* GetDetail() const
```

See *AddDetail* method.

GetName

```
std::string GetName() const
```

See *SetName* method.

GetOption

```
OEOptions* GetOption(const std::string& name)
const OEOptions* GetOption(const std::string& name) const
```

Get the sub-options with the specified name. Returns a NULL pointer if a sub-options with specified name does not exist.

GetOptions

```
OEIterBase<const OEOptions>* GetOptions(const bool recursive = true) const
OEIterBase<OEOptions>* GetOptions(const bool recursive = true)
```

Get the sub-options of this options. A value of `true` for `recursive` compiles all the sub-options from the subsequent sub-options.

GetOrderPriority

```
int GetOrderPriority() const
```

See *SetOrderPriority* method.

GetParameter

```
OEParameter* GetParameter(const std::string& name)
const OEParameter* GetParameter(const std::string& name) const
```

Get the parameter with the specified name. Returns a NULL pointer if a parameter with specified name does not exist.

GetParameters

```
OEIterBase<const OEParameter>* GetParameters(const bool recursive = true) const
OEIterBase<OEParameter>* GetParameters(const bool recursive = true)
OEIterBase<const OEParameter>* GetParameters(const std::string& name,
                                             const bool recursive = true) const
OEIterBase<OEParameter>* GetParameters(const std::string& name,
                                       const bool recursive = true)
```

Get the parameters from this options. If a name is specified only parameters with the specified name are provided. A value of `true` for `recursive` compiles all the parameters from the subsequent sub-options.

HasDupParameters

```
bool HasDupParameters(const bool visibleOnly = true) const
```

Check if a there are duplicate parameters in this options. Two parameters are considered duplicates if they have a common name. If `visibleOnly` is set to `True`, parameters with visibility *OEParamVisibility.Hidden* are ignored from this check. This operation is always recursive to the associated sub-options.

HasOption

```
bool HasOption(const std::string& name) const
```

Check if a sub-options with the specified name exists in the options.

HasParameter

```
bool HasParameter(const std::string& name) const
```

Check if a parameter with the specified name (or alias) exists in the options.

SetBrief

```
bool SetBrief(std::string brief)
```

Sets the brief description of the Options.

SetName

```
bool SetName(std::string name)
```

Sets the name of the Options.

SetOrderPriority

```
bool SetOrderPriority(int order)
```

Sets the order priority of the Options, which is used to control the order Options appear in help lists. **Default: 0**

SetParameterVisibility

```
bool SetParameterVisibility(const unsigned visibility)
bool SetParameterVisibility(const std::string& name,
                            const unsigned visibility)
```

Sets the visibility of the parameters to the specified value. This operation is always recursive to the associated sub-options. Valid settings are defined in the *OEParamVisibility* namespace.

UpdateValues

```
bool UpdateValues(const OEOptions& opts)
```

Update parameters values of this options, from the parameter values of the soecified options. Two parameters are considered same if they have a common name and data type. This operation is always recursive to the associated sub-options.

OEPODParameter

```
template<class TT>
class OEPODParameter : public OETypedParameter<TT>
```

The *OEPODParameter* is an abstract base class, derived from *OETypedParameter*. This is a templated class that provides type-specific implementation of parameters that has plain-old-data (POD) type value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*

- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

The *OEPODParameter* class defines the following public methods:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

The following classes derive from this class:

- *OEBoolParameter*
- *OEDoubleParameter*
- *OEFloatParameter*
- *OEIntParameter*
- *OEStringParameter*
- *OEUIntParameter*

GetDefaultsVector

```
std::vector<TT> GetDefaultsVector() const
```

Returns a vector of values containing the current defaults.

GetSettingsVector

```
std::vector<TT> GetSettingsVector() const
```

Returns a vector of values containing the current settings.

GetValuesVector

```
std::vector<TT> GetValuesVector() const
```

Returns a vector of values containing the current user-defined values.

OEPointerParameter

```
template<class TT>
class OEPointerParameter : public OEParameter
```

The *OEPointerParameter* is an abstract base class, derived from *OEParameter*. This is a templated class that provides type-specific implementation of parameters that has object-pointer-type value. A *OEPointerParameter* always contains a single default or user-defined value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

OEStringParameter

```
class OEStringParameter : public OEPODParameter<std::string>
```

The *OEStringParameter* represents parameter that has `string` value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*
- *GetSettingsVector*
- *GetValuesVector*

The following classes derive from this class:

- *OEMolReaderParameter*
- *OEMolWriterParameter*

Constructors

```
OEStringParameter()
OEStringParameter(const std::string& name)
OEStringParameter(const std::string& name,
```

(continues on next page)

(continued from previous page)

```
                  const std::string& default)
OEStringParameter(const OEStringParameter&)
```

Default and copy constructors.

Constructs an *OEStringParameter* instance using the specified set of parameters.

OEUIntParameter

```
class OEUIntParameter : public OEPODParameter<unsigned>
```

The *OEUIntParameter* represents parameter that has `unsigned` value.

Following methods are publicly inherited from *OEParameter*:

- *AddAlias* and *GetAliases*
- *AddDetail* and *GetDetail*
- *AddIllegalRange* and *GetIllegalRanges*
- *AddIllegalValue* and *GetIllegalValues*
- *AddLegalRange* and *GetLegalRanges*
- *AddStringDefault* , *GetStringDefault* and *GetStringDefaults*
- *AddStringValue* , *GetStringValue* and *GetStringValues*
- *ClearDefaults*
- *ClearValues*
- *CreateCopy*
- *GetBrief* and *SetBrief*
- *GetHasDefault*
- *GetHasValue*
- *GetIsList* and *SetIsList*
- *GetKeyless* and *SetKeyless*
- *GetName* and *SetName*
- *GetOrderPriority* and *SetOrderPriority*
- *GetVisibility* and *SetVisibility*
- *IsLegalString*
- *IsSet* and *IsSetToString*

Following methods are publicly inherited from *OETypedParameter*:

- *AddDefault* , *GetDefault* , *GetDefaults* and *SetDefault*
- *GetSetting* and *GetSettings*
- *AddValue* , *GetValue* , *GetValues* and *SetValue*

Following methods are publicly inherited from *OEPODParameter*:

- *GetDefaultsVector*

- *GetSettingsVector*
- *GetValuesVector*

The following classes derive from this class:

- *OEConstParameter*

Constructors

```
OEUIntParameter()
OEUIntParameter(const std::string& name)
OEUIntParameter(const std::string& name, const unsigned default)
OEUIntParameter(const OEUIntParameter&)
```

Default and copy constructors.

Constructs an *OEUIntParameter* instance using the specified set of parameters.

Attention

This API is currently available in C++ and Python.

OERecord

See also

- *OERecord* chapter

4.4.2 Preliminary OESystem Constants

OEOptsConfigureStatus

This namespace contains constants.

Success

Options configuration finished successfully.

Help

Options help requested at configuration

Failed

Options configuration failed.

Types

The *Types* namespace contains a number of OEFieldType objects. These are used as arguments to the OEMakeField function, to indicate the data type of the field to be created.

The following table shows the members of the *Types* namespace and the corresponding value types.

Types member	Type stored on OERecord
Int	OELongLong (64-bit signed int)
Float	double
String	std::string
Bool	bool
Blob	std::vector<unsigned char>
Range	OERange
Record	*OERecord*
Chem::Mol	*OEMol*
Chem::DesignUnit	*OEDesignUnit*
Chem::Grid	*OEScalarGrid*
Chem::SkewGrid	*OESkewGrid*
Chem::FingerPrint	OEFingerPrint
Chem::Surface	OESurface
Chem::ShapeQuery	OEShapeQuery
IntVec	std::vector<OELongLong>
FloatVec	std::vector<double>
StringVec	std::vector<std::string>
BoolVec	std::vector<bool>
BlobVec	std::vector<std::vector<unsigned char>>
RecordVec	std::vector<OERecord>
Chem::MolVec	std::vector<OEMol>
Chem::GridVec	std::vector<OEScalarGrid>
Chem::SkewGridVec	std::vector<OESkewGrid>
Chem::FingerPrintVec	std::vector<OEFingerPrint>
Chem::SurfaceVec	std::vector<OESurface>
Chem::ShapeQueryVec	std::vector<OEShapeQuery>

Note

The types not defined in the OESystem namespace are not available until the higher level toolkits have been initialized.

See also

- OEMakeField

4.4.3 Preliminary OESystem Functions

OEConfigureItf

```
bool OEConfigureItf(OEInterface& itf, const OEOptions& opts)
```

This method populates the specified *OEInterface* with parameters and child *OEInterface* from the specified *OEOptions*.

OEConfigureOpts

```
unsigned OEConfigureOpts(OEOptions& opts, int argc, char** argv,
                         const bool writeSettings = true)
```

This method configures the parameter settings of the specified *OEOptions* from the specified command line arguments.

opts
The interface to configure.

argc
Command line arguments count.

argv
Command line argument values.

writeSettings
If the visible parameter settings should be written in standard out.

OEUpdateOpts

```
bool OEUpdateOpts(OEOptions& opts, const OEInterface& itf)
```

This method updates the parameter settings of the specified *OEOptions* with the corresponding parameter settings of the *OEInterface*. Two parameters between the *OEOptions* and *OEInterface* are considered same if they have a common name and data type.

OEMakeMatchField

Creates an OEMatchField object to be used for adding data to, or retrieving data from, an *OERecord* object. Match fields are used mainly to retrieve data from a record when you don't know the name of a field but know its type and associated metadata.

name
The name the field will be given.

t
This specifies the type of match field to be created. This will usually come from the *Types* namespace.

meta
If this field is provided, the returned OEMatchField object will only retrieve data from fields that match this metadata.

readonly
If true, the returned match field can only be used to read data to a record, not write it.

nullable
If true, the returned field match field will allow setting of null values on a record.

See also

- OEMatchField
- *OERecord*
- OEFieldMeta

OEReadRecord

4.5 OEPlatform API

4.5.1 OEPlatform Classes

OECondition

```
class OECondition
```

This class represents *OECondition*.

Constructors

```
OECondition()
```

Default and copy constructors.

Acquire

```
void Acquire()
```

Broadcast

```
void Broadcast()
```

Release

```
void Release()
```

Signal

```
void Signal()
```

Wait

```
void Wait()
```

OEDirectoryScan

```
class OEDirectoryScan
```

This class represents *OEDirectoryScan*.

The *OEDirectoryScan* class provides a portable way of retrieving the names of all the files and subdirectories in a given directory.

Constructors

```
OEDirectoryScan(const char* dname)
```

The constructor for *OEDirectoryScan* specifies the name of the directory to scan. A NULL pointer, `(char*)0`, is taken to mean the current directory and is equivalent to `"."`.

operator bool

```
operator bool() const
```

Checks whether the *OEDirectoryScan* has been successfully opened and is still valid. A return value of `true` does not guarantee that the next call to the Next method will not return `(const char*)0`.

Close

```
void Close()
```

Closes the *OEDirectoryScan* and frees any system resources. This method is invoked automatically by the classes' destructor. Following a call to Close, operator bool will always return `false`.

Next

```
const char *Next()
```

Returns the next file name sequentially in the directory scan. This function automatically advances the directory pointer. The returned value is a pointer to the local filename, *i.e.* without its directory prefix. Upon reaching the end of the directory, this function returns `(const char*)0`, and the *OEDirectoryScan* is closed.

OEFile

```
class OEFile
```

This class represents *OEFile*.

Constructors

```
OEFile(const OEFile &)
OEFile(const std::string &="")
OEFile(const std::string &, const std::string &)
```

Default and copy constructors.

AbsPath

```
std::string AbsPath() const
```

DirContents

```
std::vector<std::string> DirContents() const
```

DirPath

```
std::string DirPath() const
```

Exists

```
bool Exists() const
```

FileName

```
std::string FileName() const
```

IsDirectory

```
bool IsDirectory() const
```

IsExecutable

```
bool IsExecutable() const
```

IsFile

```
bool IsFile() const
```

IsLink

```
bool IsLink() const
```

IsReadable

```
bool IsReadable() const
```

IsWritable

```
bool IsWritable() const
```

Refresh

```
bool Refresh()
```

Set

```
bool Set(const std::string &)
bool Set(const std::string &, const std::string &)
```

Size

```
oesize_t Size() const
```

OEHalfFloat

```
struct OEHalfFloat
```

This class can be used to store a 32-bit float as a 16-bit float as defined by the `IEEE 754-2008` standard. Useful for reducing memory consumption when most of the data is 32-bit floating point values as the expense of precision.

> **Note**
>
> This is a "storage only" class, all math should still be done as float or double and then converted to 16 bits for storage.

Constructors

```
OEHalfFloat()
OEHalfFloat(double)
OEHalfFloat(float)
OEHalfFloat(long double)
OEHalfFloat(oe_half data)
```

Default constructors initializes to the value of `0.0`. Otherwise convert the value from `float`, `double`, or `long double` and construct with that.

GetValue

```
float GetValue() const
```

Retrieve the value as a 32-bit float.

OELock

```
class OELock
```

This class can be used to ensure an *OEMutex* object is properly released when this object falls out of scope.

```
OELock  = std::lock_guard<OEMutex>;
```

See also

- *OEMutex* class
- std::lock_guard

OELockCondition

```
class OELockCondition
```

This class can be used to ensure an *OECondition* object is properly released when this object falls out of scope.

Warning

This may not prevent deadlocks as *OECondition.Signal* or *OECondition.Broadcast* still need to be explicitly called by the user.

Constructors

```
OELockCondition(OECondition& cond)
```

This constructor calls *OECondition.Acquire*. The destructor of this class will be responsible for calling *OECondition.Release*.

OELockSpinlock

```
class OELockSpinlock
```

This class can be used to ensure an *OESpinlock* object is properly released when this object falls out of scope.

```
OELockSpinlock = std::lock_guard<OESpinlock>;
```

See also

- *OESpinlock* class
- std::lock_guard

OEMutex

```
struct OEMutex
```

The *OEMutex* class provides a portable MUTual EXclusion device, that is useful for protecting shared data structures from concurrent modification.

See also

- *OELock* class
- *OETryMutex* class

Constructors

```
OEMutex()
```

Default constructor.

Acquire

```
void Acquire()
```

Acquires (locks) the *OEMutex*. If the *OEMutex* is unlocked, it is locked by this calls and becomes owned by the calling thread. If the *OEMutex* is already locked, by this or another thread, *OEMutex.Acquire* suspends the calling thread until the *OEMutex* is released.

Release

```
void Release()
```

Releases (unlocks) the *OEMutex*. The *OEMutex* object must have previously been locked by the calling thread.

OEOnce

```
template<class T>
struct OEOnce
```

OEOnce is deprecated. OpenEye strongly recommends using one of the alternatives below instead of `OEOnce`

OEOnce was created to facilitate thread-safe initialization of function-local-statics before C++11's magic-statics were universally available. Its implementation has become redundant since moving to VS2015 with full C++11 compliance. All of the existing uses will continue to work as intended without change, with minimal overhead in compile and run time for optimized release builds, in most cases.

Alternatives

- Function local static instances:
  - Only use when required - there is a performance cost!
  - Make const unless you can't.
  - Fully construct static local in a single line:
    - Via constructor: `static const SomeClass someInstance{"abc", 1.23f, 456};`
    - Via function result: `static const int tag = OEGetTag("my_tag");`
    - Use `std::unique_ptr` when you must have a pointer, or for very large objects: `static const std::unique_ptr<Class2> up{factoryFunc()};`
- When call-once is truly desired:
  - `std::atomic_flag`

        * Define at file scope if you can
        * init with = `ATOMIC_FLAG_INIT;`
        * Use `std::atomic_flag::test_and_set` in conditional
    - `std::once_flag/call_once`
        * Define `std::once_flag` at file scope if you can
        * Provides exception handling
        * Can handle more complex situations

Original Documentation

This class is used to do thread-safe one time initialization of an object as a function local static. It is intended to imitate the behavior of the C++11 "magic statics" feature. The default constructor of the class `T` is guaranteed to be called once, and only once, and all other threads will block until initialization is complete.

The following code snippet demonstrates how to use `OEOnce` to cache an initialized *OESubSearch* object to be used by subsequent function invocations:

```
class PhenylSubSearch
{
  OESubSearch _subsrch;
public:
  PhenylSubSearch() : _subsrch("c1ccccc1") { }
  const OESubSearch &GetSubSearch() const { return _subsrch; }
};

bool HasPhenyl(const OEMolBase &mol)
{
  static OEOnce<PhenylSubSearch> phenylSubSrch = OE_ONCE_INIT;
  return phenylSubSrch->GetSubSearch().SingleMatch(mol);
}
```

Warning

The thread safety of the object `T` is the responsibility of the user. `OEOnce` will only ensure that there is only one of them initialized for the whole lifetime of the process and that its destructor is called during process exit.

GetValue

```
T* GetValue()
```

Returns the pointer to a singleton object initialized in a thread safe way. The pointer will only be returned once at least one thread has fully initialized the object.

OESpinlock

```
struct OESpinlock
```

The *OESpinlock* class provides a portable mutual exclusion device, that is useful for protecting shared data structures from concurrent modification. It provides the exact same semantics as a *OEMutex* object, however, it has very different

performance semantics. Spin locks will optimistically continually try to acquire the lock in a tight loop. *OEMutex* will typically yield the processor back to the operating system. Spin locks are useful to protect very cheap operations.

See also

- *OELockSpinlock* class

Constructors

```
OESpinlock()
```

Default constructor.

Acquire

```
void Acquire()
```

Acquires (locks) the *OESpinlock*. If the *OESpinlock* is unlocked, it is locked by this thread and becomes owned by the calling thread. If the *OESpinlock* is already locked, by this or another thread, *OESpinlock.Acquire* spins in a tight loop until the *OESpinlock* is released. No attempt is made at fairness between threads.

Release

```
void Release()
```

Releases (unlocks) the *OESpinlock*.

OEThread

```
class OEThread
```

Abstracts away the notion of an operating system thread in a cross-platform way. Users can subclass this class and implement the *OEThread.Run* method in order to create new operating system threads running any arbitrary code.

Constructors

```
OEThread()
```

Default constructor.

GetArg

```
void* GetArg()
```

Used by the implementation to pass the `void *` argument from *OEThread.Start* to the *OEThread.Run* method in a new thread. Not guaranteed to return anything useful on all platforms, and may go away in a future version.

GetReturn

```
void* GetReturn()
```

Used by the implementation to pass the `void *` returned from *OEThread.Run* to the *OEThread.Join* method in a new thread. Not guaranteed to return anything useful on all platforms, and may go away in a future version. If the thread is still running, this method can data race with the running thread, this should only be called after this thread is joined.

GetStackSize

```
unsigned int GetStackSize() const
```

Return the stack size chosen for this thread by a previous call to *OEThread.SetStackSize*. By default, this method will return `0`, specifying that the operating system's default stack size will be used.

Join

```
void* Join()
```

Blocks and wait for the thread to finish, returning the `void *` returned by the overloaded *OEThread.Run* method.

Run

```
void* Run(void* targ)=0
```

Pure virtual method expected to be overloaded by the inheriting class. This method will be started in a separate operating system thread after a successful call to *OEThread.Start*.

SetArg

```
void SetArg(void* targ)
```

Used by the implementation to pass the `void *` argument from *OEThread.Start* to the *OEThread.Run* method in a new thread. Not guaranteed to return anything useful on all platforms, and may go away in a future version.

SetReturn

```
void SetReturn(void* targ)
```

Used by the implementation to pass the `void *` returned from *OEThread.Run* to the *OEThread.Join* method in a new thread. Not guaranteed to return anything useful on all platforms, and may go away in a future version.

SetStackSize

```
void SetStackSize(unsigned int stackSize)
```

Specify the maximum size a stack can grow for this newly created operating system thread. This method must be called before calling *OEThread.Start*. Depending on the platform and the operating system settings, *OEThread.Start* may fail and return `false` if the value is considered inappropriate by the operating system. The value of `0` specifies to just use the operating system default.

Start

```
bool Start(void* targ = 0)
```

Returns `true` if the operating system thread is successfully created, `false` otherwise.

OEThreadLocal

```
template<class T>
class OEThreadLocal
```

This class represents *OEThreadLocal*.

Constructors

```
OEThreadLocal()
```

Default constructor.

OETryMutex

```
class OETryMutex
```

This class represents *OETryMutex*.

The *OETryMutex* class provides a portable MUTual EXclusion device, that is useful for protecting shared data structures from concurrent modification. This version also contains *OETryMutex.Try* method that allows for a failed acquire. *OETryMutex* is slightly more costly than *OEMutex*.

Constructors

```
OETryMutex()
```

Default constructor.

Acquire

```
void Acquire()
```

Acquires (locks) the *OETryMutex*. If the *OETryMutex* is unlocked, it is locked by this calls and becomes owned by the calling thread. If the *OETryMutex* is already locked, by this or another thread, *OETryMutex.Acquire* suspends the calling thread until the *OETryMutex* is released.

Release

```
void Release()
```

Releases (unlocks) the *OETryMutex*. The *OETryMutex* object must have previously been locked by the calling thread.

Try

```
bool Try()
```

Attempts to acquire (lock) the *OETryMutex*. If the *OETryMutex* is unlocked, it is locked by this call, becomes owned by the calling thread, and returns `true`. If the *OETryMutex* is already locked, by this or another thread, *OETryMutex.Try* returns `false`.

oeifstream

```
class oeifstream : public oeistream
```

This class represents *oeifstream*.

Implements an input stream capable of reading from a specified file or from any generic data source with an associated file descriptor.

The following methods are publicly inherited from *oeistream*:

operator>>	*gettoken*	*skipbyte*
close	*open*	*skipline*
get	*peekbyte*	
getbyte	*read*	
getline	*skip*	

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeifstream()
oeifstream(int ifd)
oeifstream(const char* filename)
oeifstream(const std::string& filename)
```

Creates a new *oeifstream*. If a valid file descriptor ('ifd') is passed to the constructor, the stream will use the associated data source for input. Otherwise, the constructors expect the name of an existing file from which to read. If no parameter is passed to the constructor, the stream can later be opened using the open command defined in *oeistream* or the *oeifstream.openfd* command defined in this class.

fd

```
int fd() const
```

Returns the system dependent file descriptor associated with the stream's data source.

openfd

```
bool openfd(int ifd, bool closefd)
```

Sets the data source of the stream to the specified file descriptor 'ifd'. If 'closefd' is `false`, the underlying data source is not closed before changing to the new data source.

pread

```
oesize_t pread(char* buffer, oesize_t len, oefpos_t offset) const
```

Reads 'len' bytes into 'buffer' at the specified 'offset' in the file. Returns the number of bytes actually read. This method allows multiple threads to safely read from a single file descriptor.

> **Warning**
>
> It is undefined behavior to mix calls to 'pread' other *oeistream* read method calls on the same object.

oeigzstream

```
class oeigzstream : public oeiwrapperstream
```

This class represents *oeigzstream*.

Implements a wrapper stream capable of dynamically uncompressing gzipped data read off of the wrapped stream.

The following methods are publicly inherited from *oeiwrapperstream*:

clear	*set*	*stream*

The following methods are publicly inherited from *oeistream*:

operator>>	*gettoken*	*skipbyte*
close	*open*	*skipline*
get	*peekbyte*	
getbyte	*read*	
getline	*skip*	

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeigzstream()
oeigzstream(oeistream* is, bool del)
```

Creates a new *oeigzstream*.

> **See also**
>
> *oeiwrapperstream* for more details on the functionality of this class.

oeiofstream

```
class oeiofstream : public oeofstream
```

This class represents a file that can be both written to and read from at the same time. Note, it inherits from *oeofstream* and can generally be used as an output file stream. The major distinction being that the *oeiofstream.pread* method can be used to read data back from the file provided *oeostream.flush* has been called.

The following methods are publicly inherited from *oeofstream*:

append	*fd*	*openfd*

Constructors

```
oeiofstream()
oeiofstream(const char* filename)
oeiofstream(const std::string& filename)
```

Default and copy constructors.

pread

```
oesize_t pread(char* buffer, oesize_t len, oefpos_t offset) const
```

Reads 'len' bytes into 'buffer' at the specified 'offset' in the file. Returns the number of bytes actually read. This method allows multiple threads to safely read from a single file descriptor.

> **Warning**
>
> The `flush` method must be called before data written to this stream can be seen by the `pread` method.

oeisstream

```
class oeisstream : public oeistream
```

This class represents *oeisstream*.

Implements an input stream that uses a string of data already in memory as the input source.

The following methods are publicly inherited from *oeistream*:

operator>>	*gettoken*	*skipbyte*
close	*open*	*skipline*
get	*peekbyte*	
getbyte	*read*	
getline	*skip*	

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeisstream()
oeisstream(const oeisstream& rhs)
oeisstream(const std::string& buffer)
oeisstream(char* buffer, oesize_t len, bool copy = true)
oeisstream(const char* buffer, oesize_t len, bool copy = true)
```

Creates a new *oeisstream*. Passing any of the specified parameters to the one of the non-default constructors is equivalent to creating a new *oeisstream* with the default constructor and then calling the appropriate *oeisstream.set* method on that stream.

If the copy constructor is used, the newly created oeisstream will read from the same data string as the stream being copied from. If the 'copy' parameter was set to `true` on the stream being copied from, the new stream will create its own local copy to read from, otherwise it will just point to the same place in memory as the original stream. In addition, the newly constructed stream's reading pointer will be set to the same location as the one in the stream being copied from.

Regardless of whether or not a local copy is made, the stream takes no responsibility for the management of the original input source's memory.

clear

```
void clear()
```

Clears the association with input source associated with this stream. If the copy parameter was set to `false` when this stream was assigned or constructed, no changes will be made to the original data source.

getbuffer

```
bool getbuffer(std::string& buffer) const
```

Copies the entire contents of the input source into the specified string regardless of the current location of the reading pointer and returns whether or not this was successful.

reset

```
bool reset()
```

Resets the reading pointer to the beginning of the input stream and returns whether or not this was successful. This is equivalent to calling the *oestream.rewind* method of *oestream*.

set

```
bool set(const std::string &)
bool set(char *, oesize_t, bool)
bool set(const char *, oesize_t, bool)
```

Sets the input source of this stream based on the specified parameters. The buffer parameter specifies the data string to be used as the input source. If the `char *` constructors are used, the length of the input buffer must be specified as well as whether or not a local copy of the buffer should be made and used as the input source. If the `std::string` method is used, a local copy of the input string will be made and used as the input source.

Regardless of whether or not a local copy is made, the stream takes no responsibility for the management of the original input source's memory.

oeistdstream

```
class oeistdstream : public oeistream
```

This class represents *oeistdstream*.

Implements a wrapper stream around a standard C++ input stream (`std::istream`).

The following methods are publicly inherited from *oeistream*:

operator>>	*gettoken*	*skipbyte*
close	*open*	*skipline*
get	*peekbyte*	
getbyte	*read*	
getline	*skip*	

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeistdstream()
oeistdstream(std::istream* isptr, bool del)
```

Creates a new *oeistdstream*. Passing the 'isptr' and 'del' parameters is equivalent to using the default constructor and then calling *set(isptr, del)* on the newly created stream.

clear

```
void clear()
```

Clears the pointer to the wrapped stream. If the 'del' parameter was set to `true` when the wrapped stream was set (*oeistdstream.set*), the wrapped stream will be deleted before it is cleared.

set

```
bool set(std::istream* isptr, bool del)
```

Sets 'isptr' to be the stream wrapped by this class. If 'del' is `true`, the wrapped stream will be deleted when the clear method is called or when this class is deleted. It is important to understand that by setting 'del' to `true`, ownership of the stream's memory is transferred to this class, and if it is `false`, you retain the responsibility for managing the stream's memory.

stream

```
std::istream* stream()
```

Returns a pointer to the wrapped stream if present, otherwise `0`.

oeistream

```
class oeistream : public oestream
```

This class represents *oeistream*.

Defines the platform independent interface and implementation for a generic input data stream class.

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

The following classes derive from this class:

- *oeifstream*
- *oeigzstream*
- *oeisstream*
- *oeistdstream*
- *oeiwrapperstream*
- *oeprogressstream*

Constructors

```
oeistream()
```

Creates an instance of an *oeistream*. This class is designed to provide an interface and some functionality; however, this class does not define a data source and therefore instances of this class have no independent functionality.

close

```
void close()
```

Closes the input stream. All future attempts to read from this stream will fail until it is opened again with new input.

get

```
bool get(char& c)
bool get(signed char& c)
bool get(unsigned char& c)
```

Gets the next byte off of the stream and assigns its value to the parameter 'c'. Returns whether or not the byte read equals EOF.

getbyte

```
int getbyte()
```

Returns the next byte off of the stream.

getline

```
bool getline(std::string& sbuffer)
bool getline(char* buffer, oesize_t max)
bool getline(signed char* buffer, oesize_t max)
bool getline(std::string& sbuffer, oesize_t max)
bool getline(unsigned char* buffer, oesize_t max)
```

Retrieves the next text line from the stream and directly copies its contents into the 'buffer' parameter. The newline character is not copied with the rest of the contents, though a null terminator, '0', is placed at the end of the contents. The 'max' parameter specifies the maximum size of the memory passed in as the 'buffer' argument. If a newline character is not found within 'max - 1' characters, only the specified number of characters will be copied into the buffer and all the remaining characters in the stream before the next newline character will be discarded.

gettoken

```
bool gettoken(char* buffer, oesize_t max)
bool gettoken(signed char* buffer, oesize_t max)
bool gettoken(std::string& sbuffer, oesize_t max)
bool gettoken(unsigned char* buffer, oesize_t max)
```

Retrieves the next whitespace delimited token from the stream and directly copies its contents into the 'buffer' parameter. This function discards the whitespace characters read when searching for the next token. The 'max' parameter specifies the maximum number of characters allowed per token. If a whitespace character is not found within the specified number of characters, only the specified number of characters will be copied into the buffer and all the remaining characters in the stream before the next whitespace character will be discarded.

open

```
bool open(const char* name)
bool open(const std::string& name)
bool open(unsigned char* buffer, oesize_t len)
```

Opens the input stream with data from the specified sources. Passing a 'name' parameter will result in invocation of a protected virtual method to appropriate handle the specified input source. Passing a pointer to a character buffer along with the buffer's size will result in the stream wrapping the 'buffer' and using that as the stream data source. Returns whether or not data can now be read from the stream.

peekbyte

```
int peekbyte()
```

Returns the value of the next byte without removing that byte from the input stream. Returns -1 if the stream is not valid.

read

```
oesize_t read(std::string& sbuffer)
oesize_t read(char* buffer, oesize_t len)
oesize_t read(signed char* buffer, oesize_t len)
oesize_t read(std::string& sbuffer, oesize_t len)
oesize_t read(unsigned char* buffer, oesize_t len)
```

Reads the next len number of bytes off of the stream, stores them in the specified 'buffer' and returns the number of bytes actually read. It is the user's responsibility when using any of the three `char *` versions to make sure that the specified 'buffer' has been properly allocated and has enough room to store len bytes. This function does not add a NULL terminator to the end of the buffer.

The version that takes a `std::string` without a size will read the entire contents of the stream into the string, or until `std::bad_alloc` is thrown when the system runs out of memory.

skip

```
bool skip(oefpos_t len)
```

Discards the next number of bytes specified by 'len' in the stream. Returns whether or not there are any bytes left in the stream.

skipbyte

```
bool skipbyte()
```

Discards the next byte in the stream. Returns whether or not there are any bytes left in the stream.

skipline

```
bool skipline()
```

Discards the next text line in the stream. Returns whether or not there are any bytes left in the stream.

oeiwrapperstream

```
class oeiwrapperstream : public oeistream
```

This class represents *oeiwrapperstream*.

Implements a wrapper class to seamless contain and abstract the functionality of another *oeistream*.

The following methods are publicly inherited from *oeistream*:

operator>>	*gettoken*	*skipbyte*
close	*open*	*skipline*
get	*peekbyte*	
getbyte	*read*	
getline	*skip*	

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

The following classes derive from this class:

- *oeigzstream*
- *oeprogressstream*

Constructors

```
oeiwrapperstream()
oeiwrapperstream(oeistream* isptr, bool del)
```

Creates a new *oeiwrapperstream*. Passing the 'isptr' and 'del' parameters is equivalent to using the default constructor and then calling *set(isptr, del)* on the newly created stream.

clear

```
void clear()
```

Clears the pointer to the wrapped stream. If the 'del' parameter was set to `true` when the wrapped stream was set (*oeiwrapperstream.set*), the wrapped stream will be deleted before it is cleared.

set

```
bool set(oeistream* isptr, bool del)
```

Sets 'isptr' to be the stream wrapped by this class. If 'del' is `true`, the wrapped stream will be deleted when the clear method is called or when this class is deleted. It is important to understand that by setting 'del' to `true`, ownership of the stream's memory is transferred to this class, and if it is `false`, you retain the responsibility for managing the stream's memory.

stream

```
oeistream* stream()
```

Returns a pointer to the wrapped stream if present, otherwise `0`.

oemultiostream

```
class oemultiostream : public OEPlatform::oeostream
```

This class represents *oemultiostream*. This class can be used to write to more than one *oeostream*.

The following methods are publicly inherited from *oeostream*:

operator<	*flush*	*putbyte*
close	*open*	*write*

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oemultiostream()
```

Default constructor.

addstream

```
void addstream(oeostream* ostr, bool owned = false)
```

This method adds the stream *ostr* for output. The second parameter can be used to give ownership to the *oemultiostream* instance, which will make the instance responsible for deletion of the *oeostream*.

clear

```
void clear()
```

Clears all streams and deletes them if owned.

oeofstream

```
class oeofstream : public oeostream
```

This class represents *oeofstream*.

Implements an output stream capable of writing to a specified file or to any generic data source with an associated file descriptor.

The following methods are publicly inherited from *oeostream*:

operator<	*flush*	*putbyte*
close	*open*	*write*

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeofstream()
oeofstream(int ofd)
oeofstream(const char* filename)
oeofstream(const std::string& filename)
```

Creates a new *oeofstream*. If a valid file descriptor ('fd') is passed to the constructor, the stream will use the associated data target for output. Otherwise, the constructors expect the name of file to be written to. If the named file does not exist, it will be created. If no parameter is passed to the constructor, the stream can later be opened using the open command defined in *oeostream* or the *oeofstream.openfd* command defined in this class.

append

```
bool append(const char* filename)
bool append(const std::string& filename)
```

Opens the specified file to which data will be appended. Returns whether or not the file was successfully opened (or created as necessary).

fd

```
int fd() const
```

Returns the system dependent file descriptor associated with the stream's data target.

openfd

```
bool openfd(int ofd, bool closefd)
```

Sets the data target of the stream to the specified file descriptor 'fd'. If 'closefd' is `false`, the underlying data target is not closed before changing to the new data source.

oeogzstream

```
class oeogzstream : public oeowrapperstream
```

This class represents *oeogzstream*.

Implements a wrapper stream capable of dynamically compressing data in gzip format and then passing that along to the wrapped stream.

The following methods are publicly inherited from *oeowrapperstream*:

clear	*set*	*stream*

The following methods are publicly inherited from *oeostream*:

operator<	*flush*	*putbyte*
close	*open*	*write*

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeogzstream()
oeogzstream(oestream* os, bool del)
```

Creates a new *oeogzstream*.

See also

See *oeowrapperstream* for more details on the functionality of this class.

oeosstream

```
class oeosstream : public oestream
```

This class represents *oeosstream*.

Implements an output stream that writes into an internally contained string.

The following methods are publicly inherited from *oeostream*:

operator<	*flush*	*putbyte*
close	*open*	*write*

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeosstream()
oeosstream(const char *)
oeosstream(const unsigned char *, oesize_t)
```

Creates a new *oeosstream* which is immediately ready to be written to. Passing any of the specified parameters to one of the non-default constructors is equivalent to creating a new *oeosstream* with the default constructor and then calling the appropriate *oeosstream.set* method.

If the copy constructor is used, the entire contents of the other stream's internal string are copied into the newly constructed stream's internal string.

clear

```
void clear()
```

Clears the contents of the internal string.

set

```
bool set(const char * buffer)
bool set(const unsigned char *, oesize_t len)
```

Sets the contents of the internal string to the contents of the specified 'buffer'. If the 'len' parameter is not specified, this method will copy the contents of the specified 'buffer' until a null character is reached. The null character is not copied. Returns whether or not the contents were successfully copied.

str

```
std::string str()
```

Returns a copy of the current internal string.

oeostdstream

```
class oeostdstream : public oeostream
```

This class represents *oeostdstream*.

The following methods are publicly inherited from *oeostream*:

operator<<	*flush*	*putbyte*
close	*open*	*write*

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

Constructors

```
oeostdstream()
oeostdstream(std::ostream* osptr, bool del)
```

Default and copy constructors.

clear

```
void clear()
```

set

```
bool set(std::ostream* osptr, bool del)
```

stream

```
std::ostream* stream()
```

oeostream

```
class oeostream : public oestream
```

This class represents *oeostream*.

Defines the platform independent interface and implementation for a generic output data stream class.

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

The following classes derive from this class:

- *oeofstream*
- *oeogzstream*
- *oeosstream*
- *oeostdstream*
- *oeowrapperstream*

Constructors

```
oeostream()
```

Creates an instance of an *oeostream*. This class is designed to provide an interface and some functionality; however, this class does not define a data target and therefore writing data to an instance of this is equivalent to writing to null.

close

```
void close()
```

Flushes the remaining data in the output buffer and then closes the stream. After the stream has been closed, it is then initialized and ready to be opened. Before this stream can be reused, it must first be opened with *oeostream.open*. Any attempts to write to this stream before it has been opened will fail.

flush

```
bool flush()
```

Flushes the remaining data in the output buffer onto the stream and returns whether or not it was successful. Explicit calls to this function are only necessary when one needs to make sure that all of the data that has been stored in the output buffer is actually on the output stream at that time. The output buffer is automatically flushed when it becomes full.

open

```
bool open(const char* name)
bool open(const std::string& name)
bool open(unsigned char* buffer, oesize_t len)
```

Opens the output stream to the specified sources. Passing a 'name' parameter will result in invocation of a protected virtual method to appropriate handle the specified output source. Passing a pointer to a character buffer along with the buffer's size will result in the stream wrapping the 'buffer' and using that as the stream data destination. Returns whether or not data can now be written to the stream.

putbyte

```
bool putbyte(char c)
```

Pushes the specified byte 'c' onto the output stream and returns whether or not this was successful.

write

```
oesize_t write(const char* buffer, oesize_t len)
oesize_t write(const std::string& buffer, oesize_t len)
oesize_t write(const signed char* buffer, oesize_t len)
oesize_t write(const unsigned char* buffer, oesize_t len)
```

Writes 'len' number of bytes from the specified data source ('buffer') onto the output stream and returns the number of bytes that were able to be written.

oeowrapperstream

```
class oeowrapperstream : public oeostream
```

This class represents *oeowrapperstream*.

Implements a wrapper class to seamless contain and abstract the functionality of another *oeostream*

The following methods are publicly inherited from *oeostream*:

operator<	*flush*	*putbyte*
close	*open*	*write*

The following methods are publicly inherited from *oestream*:

operator bool	*rewind*	*tell*
eof	*seek*	
length	*size*	

The following classes derive from this class:

- *oeogzstream*

Constructors

```
oeowrapperstream()
oeowrapperstream(oeostream* osptr, bool del)
```

Creates a new *oeowrapperstream*. Passing the 'osptr' and 'del' parameters is equivalent to using the default constructor and then calling *set(osptr, del)* on the newly created stream.

clear

```
void clear()
```

Clears the pointer to the wrapped stream. If the 'del' parameter was set to `true` when the wrapped stream was set (*oeowrapperstream.set*), the wrapped stream will be deleted before it is cleared.

set

```
bool set(oeostream* osptr, bool del)
```

Sets 'osptr' to be the stream wrapped by this class. If 'del' is `true`, the wrapped stream will be deleted when the clear method is called or when this class is deleted. It is important to understand that by setting 'del' to `true`, ownership of the stream's memory is transferred to this class, and if it is `false`, you retain the responsibility for managing the stream's memory.

stream

```
oeostream* stream()
```

Returns a pointer to the wrapped stream if present, otherwise `0`.

oestream

```
class oestream
```

This class represents *oestream*.

The *oestream* class is the base class which sets up the platform independent interface for streaming any type of data.

The following classes derive from this class:

- *oeifstream*
- *oeigzstream*
- *oeisstream*
- *oeistdstream*

- *oeistream*
- *oeiwrapperstream*
- *oeofstream*
- *oeogzstream*
- *oeosstream*
- *oeostdstream*
- *oeostream*
- *oeowrapperstream*
- *oeprogressstream*

Constructors

```
oestream()
```

Creates an instance of an *oestream*. This class exists solely to provide an interface, therefore, instances of this class have no independently meaningful functionality.

operator bool

```
operator bool() const
```

Evaluates the stream to a Boolean value based on whether the end of the stream has been reached. This is equivalent to `!eof()`.

eof

```
bool eof() const
```

Returns whether the end of the stream has been reached.

length

```
oefpos_t length()
```

Returns the number of bytes that have been read from or written to the stream in the last *read* or *write* call.

rewind

```
bool rewind()
```

Sets the data pointer position in the stream to the beginning of the stream.

seek

```
bool seek(oefpos_t pos)
bool seek(oefpos_t offset, seek_dir dir)
```

Sets the current position of the data pointer. This is accomplished by specifying the desired position or by specifying an offset distance and a directionality from the current position. Directionality is specified using the following enumeration:

```
enum seek_dir { beg = 0, cur = 1, end = 2 };
```

size

```
oefpos_t size()
```

If possible, returns the size of the stream in bytes. For *oeostreams* this is always `zero`.

> **Note**
>
> For *oeistreams* this may not be possible to calculate without reading, *e.g.*, when the stream is a socket or pipe. In this case the implementation may return `0`.

tell

```
oefpos_t tell() const
```

Returns the current position of the data pointer in the stream.

4.5.2 OEPlatform Constants

OEPlatform

This namespace contains constants.

oein

This is a globally available input stream that reads directly from stdin.

oeout

This is a globally available output stream that writes directly to stdout.

oeerr

This is a globally available output stream that writes directly to stderr.

oenul

This is a globally available output stream that writes directly to null.

4.5.3 OEPlatform Functions

OEAddLicenseData

```
void OEAddLicenseData(const std::string & data)
```

Parses the string passed as `data` as an OpenEye license file and adds all the license keys found to the current running process.

OEAddLicenseFromHttp

```
void OEAddLicenseFromHttp(System.Uri uri)
```

License the current process with a license file retrieved from an HTTP or HTTPS request. The following caveats apply:

- There is no proxy support.
- The request has a 10 second time out.
- Content-type must be any of the followings: `text/plain`, `text/html; charset=utf-8` or `text/plain; charset=utf-8`
- An exception will be thrown if the server does not return status 200 (success) with the contents of a valid `oe_license.txt` file.

An example of setting the license:

```
        OEChem.OEAddLicenseFromHttp(new Uri("http://localhost:8000/oe_license.txt"));
```

See also

An example of a python http license server is available here: https://github.com/oess/HttpLicenseServer

OEAddLicenseKey

```
int OEAddLicenseKey(const char* key, const char* licensee,
                    const char* site)
```

Add a license to the current process to allow more OpenEye software to be used.

OEBigEndian

```
constexpr bool OEBigEndian()
```

OECloseIStream

```
void OECloseIStream(oeistream *)
```

Closes the specified input *oeistream* stream. This function only works on *oeistreams* created with the *OEOpenIStream* function and is also the only way in which they should be closed.

OECloseOStream

```
void OECloseOStream(oeostream *)
```

Closes the specified output *oeostream* stream. This function only works on *oeostreams* created with the *OEOpenOStream* function and is also the only way in which they should be closed.

OECompressionAvailable

```
bool OECompressionAvailable(const char *="gz")
```

Returns whether or not compress/uncompress functionality is available for the specified compression format.

OECompress

```
bool OECompress(std::string& dst, const std::string& src,
                const char* fmt = "gz")
bool OECompress(std::string& dst, const char* src, size_t size,
                const char* fmt = "gz")
```

Compresses the data in the original data buffer, `src`, using the specified compression format, `fmt`, and stores the compressed data in the compressed string `dst`. Returns whether or not the specified compression format was available and if the data was successfully compressed using that format.

OECreateDirectory

```
bool OECreateDirectory(const char* dname)
```

Provides a portable way of creating a new subdirectory. Upon success, this function returns the value `true`.

OECurrentTime

This function is only available in C++.

oeendl

```
oeostream& oeendl(oeostream &)
```

Pushes a newline character onto the specified output stream and then flushes that stream.

oeends

```
oeostream& oeends(oeostream &)
```

Pushes a null character onto the specified output stream.

oeflush

```
oeostream& oeflush(oeostream &)
```

Flushes the specified output stream.

OEFileCorrectSeparators

```
void OEFileCorrectSeparators(std::string& path)
```

OEFileCreate

```
OEFile OEFileCreate(const std::string& fname, int permissions = 00666)
```

OEFileDelete

```
bool OEFileDelete(OEFile& file)
bool OEFileDelete(const std::string& fname)
```

OEFileDetermineName

```
std::string OEFileDetermineName(const std::string& fname)
```

OEFileDeterminePathAndName

```
void OEFileDeterminePathAndName(const std::string& fname,
                                std::string& path, std::string& name)
```

OEFileDeterminePath

```
std::string OEFileDeterminePath(const std::string& fname)
```

OEFileExists

```
bool OEFileExists(const std::string& fname)
```

OEFileIsDirectory

```
bool OEFileIsDirectory(const std::string& fname)
```

OEFileIsFile

```
bool OEFileIsFile(const std::string& fname)
```

OEFileIsLink

```
bool OEFileIsLink(const std::string& fname)
```

OEFileIsReadable

```
bool OEFileIsReadable(const std::string& fname)
```

OEFileIsWritable

```
bool OEFileIsWritable(const std::string& fname)
```

OEFileRandomName

```
std::string OEFileRandomName()
```

> **Warning**
>
> This function is deprecated. Two threads or processes can end up with the same file name.

> **See also**
>
> Use the *OEFileTempDescriptor* function instead.

OEFileSize

```
oesize_t OEFileSize(const std::string& fname)
```

OEFileTempDescriptor

```
int OEFileTempDescriptor(std::string& fname)
```

Returns a file descriptor to a temporary file opened exclusive access for both reading and writing. The name of the file is returned in the `fname` argument string. The file name is guaranteed not to race and collide with any other thread or process on the system. It is the caller's responsibility to close the file descriptor and delete the file.

OEFileTempDirectory

```
std::string OEFileTempDirectory(const std::string& base_dir = "")
```

Create and return the name of a new temporary directory. This directory is guaranteed to be unique and free of race conditions. If no argument is given or `base_dir` is an empty string, create the directory inside the *OEFileTempPath* directory. Otherwise, create the directory inside `base_dir` given. If this function fails, an empty string it returned.

OEFileTempPath

```
std::string OEFileTempPath()
```

Returns a system dependent directory for storing temporary files.

On Linux or macOS this defaults to `/tmp`, unless the `TMPDIR` environment variable is specified, in which case that directory is used if it exists.

On Windows, this defaults to `C:\\Temp` and then `C:\\Tmp` if that doesn't exist. If the `TEMP` or `TMP` environment variables are set and point to a valid directory, that directory is used instead.

OEGetAbsolutePath

```
std::string OEGetAbsolutePath(const std::string& filename)
```

Returns the full absolute pathname to the input file or the empty string if there was an error.

OEGetAllocaThreshold

```
oesize_t OEGetAllocaThreshold()
```

Returns the maximum number of bytes that can be allocated on the stack at once.

See also

OESetAllocaThreshold function

OEGetCurrentWorkingDirectory

```
std::string OEGetCurrentWorkingDirectory()
```

Return the current working directory for the process.

OEGetDomainName

```
char* OEGetDomainName()
```

Returns the domain name of the machine currently being run on. This function is thread safe and expensive on the first call. The returned pointer does not need to be freed.

OEGetHostIdent

```
unsigned int OEGetHostIdent()
```

Returns a unique number derived from the MAC address to represent this machine uniquely.

OEGetHostName

```
char* OEGetHostName()
```

Returns the hostname of the current machine. This function is thread safe and expensive on the first call. The returned pointer does not need to be freed.

OEGetIPAddress

```
unsigned char* OEGetIPAddress()
```

Returns the IPv4 address of the machine currently being run on.

OEGetMachineName

```
std::string OEGetMachineName()
```

Returns the NetBIOS name of the local computer on Windows. Returns an empty string on other platforms.

OEGetNumProcessors

```
unsigned int OEGetNumProcessors()
```

Returns the number of processor cores available in the current machine.

OEGetPlatformInfo

```
std::string OEGetPlatformInfo()
```

Returns a description of the platform being run on, .e.g, version of Windows, or flavor of Linux distribution.

OEGetProcessMemUsage

```
void OEGetProcessMemUsage(size_t& vm_usage, size_t& resident_set)
```

Returns the total amount of virtual memory used as the `vm_usage` parameter, and the total amount of memory resident in physical memory as the `resident_set` parameter.

OEGetProcessorInfo

```
std::string OEGetProcessorInfo()
```

OEGetTimeOfDay

This function is only available in C++.

OEGetUserAppDataDir

```
std::string OEGetUserAppDataDir()
```

OEGetUserDocumentsDir

```
std::string OEGetUserDocumentsDir()
```

OEGetUserHomeDir

```
std::string OEGetUserHomeDir()
```

OEGetUserName

```
std::string OEGetUserName()
```

Returns the user name of the current user. The current implementation returns the value `User` or `Windows User` if the user name could not be determined. Future implementations of this function may return an empty string upon failure.

OEIsNaN

```
bool OEIsNaN(float f)
bool OEIsNaN(double d)
```

Returns `true` if the number passed is one of the following non-real numbers: signaling NaN, quiet NaN, negative infinity, or positive infinity.

See also

The reverse of `OEIsNaN` is *OEIsNormalNum*.

OEIsNormalNum

```
bool OEIsNormalNum(float f)
bool OEIsNormalNum(double d)
```

Returns `true` if the number passed is one of the following real numbers: negative or positive denormalized; negative or positive zero; negative or positive normalized.

See also

The reverse of `OEIsNormalNum` is *OEIsNaN*.

OELocalTime

This function is only available in C++.

OEOpenIStream

```
oeistream* OEOpenIStream(const char *)
```

Creates and returns an *oeistream* input stream capable of reading the input specified by the source parameter. The source parameter is initially parsed to determine whether a colon delimited protocol was specified. If no protocol was specified, it is assumed that the source parameter refers to a filename. If a protocol was specified, the function checks to see whether an input stream allocator and deallocator have been registered to handle the appropriate protocol. If the specified protocol has been registered, the appropriate input stream will be created and returned, otherwise the function will return `0`. Allocators and deallocators for specific protocols can be registered using the *OERegisterIStream* function.

```
oeistream *is = OEOpenIStream("pipe: echo \"hello world\"");
```

In addition, this function will attempt to assess whether or not the input data is compressed. If the data is compressed in a supported format, the appropriate decompression wrapper stream will be used to dynamically decompress the data.

Warning

All streams created with this function MUST be closed with the *OECloseIStream* function.

OEOpenOStream

```
oeostream* OEOpenOStream(const char *)
```

Creates and returns an *oeostream* output stream capable of writing to the specified target parameter. The target parameter is initially parsed to determine whether a colon delimited protocol was specified. If no protocol was specified, it is assumed that the target parameter refers to a filename. If a protocol was specified, the function checks to see whether an output stream allocator and deallocator have been registered to handle to the appropriate protocol. If the specified protocol has been registered, the appropriate output stream will be created and returned, otherwise the function will return `0`. Allocators and deallocators for specific protocols can be registered using the *OERegisterOStream* function.

```
oeostream *os = OEOpenOStream("pipe: uniq > output.txt");
```

In addition, this function will attempt to assess whether or not the output data target expects compressed data. If the expected format is supported, the appropriate compression wrapper stream will be used to dynamically compress the data.

> **Warning**
>
> All streams created with this function MUST be closed with the *OECloseOStream* function.

OEProcessFileName

```
std::string OEProcessFileName(const char *)
```

OEProcessSystemCommand

```
std::string OEProcessSystemCommand(const char *)
```

OERegisterIStream

```
bool OERegisterIStream(const char *, oeistream_alloc_t,
                       oeistream_dealloc_t)
```

OERegisterOStream

```
bool OERegisterOStream(const char *, oeostream_alloc_t,
                       oeostream_dealloc_t)
```

OERenameFile

```
bool OERenameFile(const char* oldfname, const char* newfname)
```

Renames a file portably. The 'oldfname' argument denotes the original file name, which is renamed to the filename specified by 'newfname'. This function returns `true` upon success.

OESetAllocaThreshold

```
void OESetAllocaThreshold(oesize_t size)
```

Sets the maximum number of bytes that can be allocated on the stack at once by alloca. If larger memory is requested than the set limit, then the memory will be allocated on the heap.

> **Note**
>
> The default limit is 4096. Please only reduce the limit if necessary. Reducing the default limit can have significant performance impact.

See also

OEGetAllocaThreshold function

OESetLicenseFile

```
void OESetLicenseFile(const char* fname)
```

Set the OpenEye license file to use to perform license checks. This can be used instead setting the `OE_LICENSE` environment variable.

OESleep

This function is only available in C++.

OEToolkitsGetRelease

```
std::string OEToolkitsGetRelease()
```

Returns the release name of the overall toolkits being used. This returns a string of the form `YYYY.Mon.V`, where

- `YYYY` is the four-digit year
- `Mon` is a month (e.g. Jan, Feb, etc.)
- `V` is a single digit version number (1-9)

See also

- Example program *OEChemInfo.cs*

OEUncompress

```
bool OEUncompress(std::string& dst, const std::string& src,
                  const char* fmt = "gz")
bool OEUncompress(std::string& dst, const char* src, size_t size,
                  const char* fmt = "gz")
```

Uncompresses the data in the compressed data buffer, `src`, using the specified compression format, `fmt`, and stores the uncompressed data in the uncompressed string `dst`. Returns whether or not the specified compression format was available and if the data was successfully uncompressed using that format.

OEUnlinkDirectory

```
bool OEUnlinkDirectory(const char* dname)
```

Deletes or renames a directory portably. All files and subdirectories of the given directory should be deleted before calling this function. This function returns `true` upon success.

OEUnlink

```
bool OEUnlink(const char* fname)
```

Deletes or renames a file portably. Deleting a file that is still open by a process results in implementation defined behavior. This function returns `true` upon success.

4.6 OEMath API

4.6.1 OEMath Classes

OEBox

```
class OEBox
```

OEBox class represents a box aligned to the x, y and z coordinate axis.

Constructors

```
OEBox()
OEBox(const OEBox&)
```

Default and copy constructors

Extend

```
bool Extend(const float x, const float y, const float z)
bool Extend(const float value)
```

Extend the box in all dimensions with the specified values. The box is extended simultaneously in both maximum and minimum coordinate directions

GetArea

```
float GetArea() const
```

Returns the surface area of the box

GetVolume

```
float GetVolume() const
```

Returns the volume of the box

GetXDim

```
float GetXDim() const
```

Returns the X dimension value of the box

GetYDim

```
float GetYDim() const
```

Returns the Y dimension value of the box

GetZDim

```
float GetZDim() const
```

Returns the Z dimension value of the box

GetXMax

```
float GetXMax() const
```

Returns the maximum X value of the box

GetYMax

```
float GetYMax() const
```

Returns the maximum Y value of the box

GetZMax

```
float GetZMax() const
```

Returns the maximum Z value of the box

GetXMid

```
float GetXMid() const
```

Returns the midpoint of X dimension of the box

GetYMid

```
float GetYMid() const
```

Returns the midpoint of Y dimension of the box

GetZMid

```
float GetZMid() const
```

Returns the midpoint of Z dimension of the box

GetXMin

```
float GetXMin() const
```

Returns the minimum X value of the box

GetYMin

```
float GetYMin() const
```

Returns the minimum Y value of the box

GetZMin

```
float GetZMin() const
```

Returns the minimum Z value of the box

IsInside

```
bool IsInside(const float x, const float y, const float z) const
```

Returns if the specified point is inside the box

Setup

```
bool Setup(const float x1, const float y1, const float z1,
           const float x2, const float y2, const float z2)
bool Setup(const float* xyz, const unsigned numPoints,
           const bool addbox = 0.0f)
bool Setup(const float* center, const float* extents)
```

Sets up the box.

The first overload expects the maximum and minimum values in all directions. The max and min X values are the max and min of *x1*, *x2* respectively. Minimum and maximum values of Y and Z are selected in an analogous fashion.

The second overload creates a box that encapsulates all the specified data points. The number of values in `xyz` must be three times that specified in `numPoints`. The value in `addbox` extends the box in all three dimensions with the specified value. The box is extended simultaneously in both maximum and minimum coordinate directions, when specified.

The third overload creates a box with the specified center and extent.

OEKernelPLS

```
class OEKernelPLS
```

The *OEKernelPLS* can be used to build models using kernel partial least squares technique (kPLS).

To train a model using kPLS, the descriptors are expected to be in the dot-kernel space. Kernel partial least squares (PLS) has been particularly popular in chemometrics, due to its sub-cubic runtime for learning, and an iterative construction of directions which are relevant for predicting the outputs.

The *OEKernelPLS* class defines the following public methods:

- *Fit*

- *GetB0*
- *GetBValues*
- *GetNumFeaturesUsed*
- *Predict*

Constructor

```
OEKernelPLS()
OEKernelPLS(const OEKernelPLS&)
```

Default and copy constructors.

Fit

```
bool Fit(const OESquareMatrix& kernel,
         const std::vector<double>& vecResponse,
         const unsigned maxFeatures)
```

Fit model using the provided kernel descriptor matrix.

kernel
kernel descriptor matrix.

vecResponse
vector of response corresponding to descriptors.

maxFeatures
Maximum number of PLS features to use for model fitting. A value of *0 (zero)* corresponds to choosing number of features to fit the best model that minimizes error on the training set.

GetB0

```
double GetB0() const
```

Get the fitted model intercept.

GetBValues

```
const std::vector<double>& GetBValues() const
```

Get the fitted model regression coefficients.

GetNumFeaturesUsed

```
unsigned GetNumFeaturesUsed() const
```

Returns the actual number of PLS features used for model fitting.

Predict

```
double Predict(const std::vector<double>& kernel) const
```

Returns predicted estimation for the input descriptor vector.

OEMatrix

```
class OEMatrix
```

OEMatrix class represents a Matrix.

The *OEMatrix* class defines the following public methods:

- *AddRow*
- *Clear*
- *GetData*
- *GetNumColumns*
- *GetNumRows*
- *Resize*
- *SetRowData*
- *Transpose*

Constructors

```
OEMatrix()
OEMatrix(const unsigned numRow, const unsigned numCol,
         const double initialValue = 0.0)
OEMatrix(const OEMatrix&)
```

Default and copy constructors.

The default constructor creates an empty matrix. The second overload creates a matrix with *numRows* rows and *numCols* columns, with all the values set to zero. The third overload creates a matrix with *numRows* rows and *numCols* columns, with all the values set to *initialValue*.

AddRow

```
void AddRow(const double initialValue = 0.0)
```

Add a new row to the matrix. The new row is initialized with *zero*, unless *initialValue* is provided.

Clear

```
void Clear()
```

Clear the matrix.

GetData

```
const std::vector<double>& GetData() const
```

Get the contents of the matrix. The data is organized sequentially from the first row to the last.

GetNumColumns

```
unsigned GetNumColumns() const
```

Returns the number of columns in the matrix.

GetNumRows

```
unsigned GetNumRows() const
```

Returns the number of rows in the matrix.

Resize

```
void Resize(const unsigned numRow, const unsigned numCol,
            const double initialValue = 0.0)
```

Resize the matrix to *numRows* rows and *numCols* columns. The matrix is initialized with *zero*, unless *initialValue* is provided.

SetRowData

```
void SetRowData(const unsigned iRow,
                const std::vector<double>& vecValues)
```

Set *vecValues* as content of the *iRow*-th row.

Transpose

```
OEMatrix Transpose() const
```

Returns the transpose of the matrix.

OESphere

OESphere class represents a sphere.

Constructor

```
OESphere()
OESphere& operator=(const OESphere&)
OESphere(const float x, const float y, const float z,
         const float rad)
```

Default and copy constructors. The third overload requires the center of the sphere, specified by *x, y, and z*, and the radius.

Assignment operator.

GetArea

```
float GetArea() const
```

Returns the surface area of the sphere

GetVolume

```
float GetVolume() const
```

Returns the volume of the sphere

GetX

```
float GetX() const
```

Returns the X coordinate of the center of the sphere

GetY

```
float GetY() const
```

Returns the Y coordinate of the center of the sphere

GetZ

```
float GetZ() const
```

Return the Z coordinate of the center of the sphere

GetRad

```
float GetRad() const
```

Returns the radius of the sphere

IsInside

```
bool IsInside(const float x, const float y, const float z) const
```

Returns if the specified point is inside the sphere

SetRad

```
bool SetRad(const float rad)
```

Sets the radius of the sphere to *rad*. Will fail and return false if *rad* < 0.

SetCenter

```
bool SetCenter(const float x, const float y, const float z)
```

Sets the center of the sphere.

OESquareMatrix

```
class OESquareMatrix : public OEMatrix
```

OESquareMatrix class represents a square matrix.

Following methods are publicly inherited from *OEMatrix*:

- *Clear*
- *GetData*
- *GetNumColumns*
- *GetNumRows*
- *SetRowData*
- *Transpose*

The *OESquareMatrix* class defines the following public methods:

- *Resize*

Constructors

```
OESquareMatrix()
OESquareMatrix(const unsigned numRows,
               const double initialValue = 0.0)
OESquareMatrix(const OESquareMatrix&)
```

Default and copy constructors.

The default constructor creates an empty matrix. The second overload creates a matrix with *numRows* rows and columns, with all the values set to zero. The third overload creates a matrix with *numRows* rows and columns, with all the values set to *initialValue*.

Resize

```
void Resize(const unsigned numRow, const double initialValue = 0.0)
```

Resize the matrix to *numRows* rows and columns. The matrix is initialized with *zero*, unless *initialValue* is provided.

4.6.2 OEMath Constants

OEMath

This namespace contains constants.

Rad2Deg

Conversion coefficient for angles from radians to degrees.

Deg2Rad

Conversion coefficient for angles from degrees to radians.

Pi

Numerical constant pi, π, the circumference of a circle divided by its diameter.

Pi2

Twice the numerical value of pi, π. This is the number of radians in a complete revolution.

OEConst

This namespace contains constants.

c

The speed of light in $meters/second$.

h

The Planck's constant in $J * s$ (joule-seconds).

hbar

The reduced Planck constant which is equal to the `Planck constant` divided by (or reduced by) 2π.

NA

The Avogadro constant in mol^{-1}.

k

Boltzmann constant in $JK^{-1}mol^{-1}$.

e

The elementary charge in $coulombs$.

R

The gas constant in JK^{-1}.

me

The electron mass constant in kg.

mp

The proton mass constant in kg.

mu

The atomic mass constant in kg.

eps0

The electric constant (vacuum permittivity) in $C^2 * N^{-1} * m^{-2}$

cal2J

The thermodynamic calorie to Joule factor.

4.6.3 OEMath Functions

OEBoxArea

```
float OEBoxArea(const OEBox& box)
```

Return the area of the box.

OEBoxExtend

```
bool OEBoxExtend(OEBox& box, const float x_ext, const float y_ext,
                 const float z_ext)
```

Extends each side of the *box* by the specified amount. *x_ext* extends the two faces of the box normal to the x-axis. Thus the x dimension of the box will increase by 2 times *x_ext*. The *y_ext* and *z_ext* perform the same function for the y and z faces respectively.

```
bool OEBoxExtend(OEBox& box, const float ext)
```

Extends every face of *box* by *ext*. Thus every dimension of the box will increase by 2 times *ext*.

OEBoxTranslate

```
bool OEBoxTranslate(OEBox& box, const float x_trans,
                    const float y_trans, const float z_trans)
```

Translates *box* a distance specified by (*x_trans*, *y_trans*, *z_trans*)

OEBoxVolume

```
float OEBoxVolume(const OEBox& box)
```

Return the volume of the box

OEBoxXDim

```
float OEBoxXDim(const OEBox& box)
```

Returns the X dimension of the box.

OEBoxXMid

```
float OEBoxXMid(const OEBox& box)
```

Returns the X midpoint of the box.

OEBoxYDim

```
float OEBoxYDim(const OEBox& box)
```

Returns the Y dimension of the box.

OEBoxYMid

```
float OEBoxYMid(const OEBox& box)
```

Returns the Y midpoint of the box.

OEBoxZDim

```
float OEBoxZDim(const OEBox& box)
```

Returns the Z dimension of the box.

OEBoxZMid

```
float OEBoxZMid(const OEBox& box)
```

Returns the Z midpoint of the box.

OECenter

```
template<class T, class U>
void OECenter(T* c, unsigned int size, U* trans)
```

Moves the cartesian points specified by `c` so that the collective center-of-mass is located at the origin, `(0.0, 0.0, 0.0)`. The translation applied to every point is returned as the `trans` parameter.

OEGeom2DAdd

```
void OEGeom2DAdd(float[] x, float[] p)
void OEGeom2DAdd(double[] x, double[] p)
void OEGeom2DAdd(float[] x, float[] p, float[] q)
void OEGeom2DAdd(double[] x, double[] p, double[] q)
```

Two dimensional vector addition. The first form computes $x+ = p$ and the second form computes $x = p + q$.

OEGeom2DAngle

```
template<class T>
double OEGeom2DAngle(const T* p, const T* q, const T* r)
```

Determines the angle in two dimensions between `p`, `q` and `r`. The result is in radians. The two dimensional array `q` represents the vertex for angle calculation.

OEGeom2DCopy

```
template<class T>
void OEGeom2DCopy(T* x, const T* p)
```

Two dimensional vector assignment. The vector `x` is assigned the value of `p`.

x
: Output two dimensional array.

p
: Input two dimensional array.

OEGeom2DDistance

```
template<class T>
double OEGeom2DDistance(const T* p, const T* q)
```

Determines the distance in two dimensions between `p` and `q`.

p
: First two dimensional input array.

q
: Second two dimensional input array.

OEGeom2DDistance2

```
template<class T>
double OEGeom2DDistance2(const T* p, const T* q)
```

Determines the squared distance in two dimensions between `p` and `q`. When it is possible to use the square of the distance, this is more efficient than using *OEGeom2DDistance*.

p
: First two dimensional input array.

q
Second two dimensional input array.

OEGeom2DDotProd

```
template<class T>
double OEGeom2DDotProd(const T* p, const T* q)
```

Determines the dot-product in two dimensions between p and q.

OEGeom2DMagnitude

```
template<class T>
double OEGeom2DMagnitude(T* x)
```

Returns the length of the two element vector x.

x
Input two element vector.

OEGeom2DMidpoint

```
template<class T>
void OEGeom2DMidpoint(T* x, const T* p, const T* q)
```

Determines the midpoint x between p and q in two dimensions.

x
Output two dimensional array.

p
Input two dimensional array.

q
Input two dimensional array.

OEGeom2DNormalize

```
template<class T>
void OEGeom2DNormalize(T* x)
```

Normalizes the two dimensional vector x to a unit vector.

x
Two dimensional array to normalize.

OEGeom2DRotate

```
template<class T, class U>
void OEGeom2DRotate(T* xyz, const U* m, unsigned int ncoord)
```

Rotates the two dimensional coordinates by the 2×2 rotation matrix passed as the argument m. For ncoord sets of coordinates the input array xyz is expected to have a length of $2 * ncoord$.

xyz
Array of coordinates to rotate

m
Rotation matrix

ncoord
Number of coordinate sets

OEGeom2DScale

```
template<class T>
void OEGeom2DScale(T* x, const T p)
template<class T>
void OEGeom2DScale(T* x, const T p, const T* q)
```

Two dimensional vector scaling. The first form computes $x* = p$ and the second form computes $x = p * q$.

OEGeom2DSetLength

```
template<class T>
void OEGeom2DSetLength(T* x, T len)
```

Sets the two dimensional vector `x` to have a magnitude of `len`.

OEGeom2DSubtract

```
template<class T>
void OEGeom2DSubtract(T* x, const T* p)
template<class T>
void OEGeom2DSubtract(T* x, const T* p, const T* q)
```

Two dimensional vector subtraction. The first form computes $x- = p$ and the second form computes $x = p - q$.

OEGeom2DTranslate

```
template<class T, class U>
void OEGeom2DTranslate(T* xyz, const U* t, unsigned int ncoord)
```

Translates the two dimensional coordinates by the vector passed as the argument `t`. The vector `t` must be of size two and contain the Cartesian translation vectors. For `ncoord` sets of coordinates the input array `xyz` is expected to have a length of $2 * ncoord$.

xyz
Array of coordinates to translate

t
Translation vector array

ncoord
Number of coordinate sets

OEGeom3DAbsTorsion

```
template<class T>
double OEGeom3DAbsTorsion(const T* p, const T* q, const T* r,
                          const T* s)
```

Determines the absolute value of the torsion between three dimensional coordinates `p`, `q`, `r` and `s`. The result is in radians.

OEGeom3DAdd

```
template<class T, class U>
void OEGeom3DAdd(T* x, const U* p)
template<class T>
void OEGeom3DAdd(T* x, const T* p, const T* q)
```

Three dimensional vector addition. The first form computes $x+ = p$ and the second form computes $x = p + q$.

OEGeom3DAllZero

```
template<class T>
bool OEGeom3DAllZero(const T* x, unsigned int ncoord)
```

Returns `true` if all the coordinates pointed to by `x` are zero.

x
: Three dimensional coordinates array to check.

ncoord
: Number of coordinates in the input array.

OEGeom3DAngle

```
template<class T>
double OEGeom3DAngle(const T* p, const T* q)
template<class T, class U, class V>
double OEGeom3DAngle(const T* p, const U* q, const V* r)
```

Determines the angle in three dimensions. The first form is between two vectors `p` and `q`, and the second form is between three cartesian coordinate points `p`, `q` and `r` with `q` being the vertex. The result is in radians.

OEGeom3DAngleCoord

```
template<class T>
void OEGeom3DAngleCoord(T* dst, const T* center, const T* r1, T ang,
                        T len)
template<class T>
void OEGeom3DAngleCoord(T* dst, const T* center, const T* r1,
                        const T* n, T ang, T len)
```

Calculates cartesian coordinate point `dst` that has a distance of `len` from the point `center`, and the vector $(dst - center)$ forms an angle of `ang` radians with the vector $(r1 - center)$. The second form places it on (or to the closest possible to) the plane defined by the normal `n`. The first form chooses the normal to be either the `x` or the `y` coordinate based on the proximity of $(r1 - center)$ to these axes.

OEGeom3DCopy

```
template<class T>
void OEGeom3DCopy(T* x, const T* p)
```

Three dimensional vector assignment. The vector 'x' is assigned the value of 'p'.

x
: Output two dimensional array.

p
: Input two dimensional array.

OEGeom3DCrossProd

```
template<class T>
void OEGeom3DCrossProd(T* x, const T* p, const T* q, const T* r)
template<class T, class U, class V>
void OEGeom3DCrossProd(T* x, const U* p, const V* q)
```

Calculates the cross product of the two three element vectors `p` and `q`. The first form computes $x = (p - q) * (r - q)$ and the second form computes $x = p * q$.

OEGeom3DDistance

```
template<class T, class U>
double OEGeom3DDistance(const T* p, const U* q)
```

Returns the Euclidean distance between the two cartesian points, `p` and `q`.

OEGeom3DDistance2

```
template<class T, class U>
double OEGeom3DDistance2(const T* p, const U* q)
```

Returns the square of the Euclidean distance between the two cartesian points, `p` and `q`. When it is possible to use the square of the distance, this is more efficient than using *OEGeom3DDistance*.

OEGeom3DDotProd

```
template<class T, class U>
double OEGeom3DDotProd(const T* p, const U* q)
```

Returns the dot product of the two given three element vectors `p` and `q`.

OEGeom3DEulerRotate

```
template<class T, class U>
void OEGeom3DEulerRotate(T* xyz, const U* angles, unsigned int ncoord)
```

Rotates the coordinates `xyz` by using the three Eulerian angles given by `angles`. The number of coordinates in `xyz` is given by `ncoord`.

OEGeom3DEulerToQuaternion

```
template<class T, class U>
void OEGeom3DEulerToQuaternion(T* quat, const U* angles)
```

Constructs the four element quaternion `quat` for the rotation given by the three Euler angles `angles`.

OEGeom3DEulerToRotMatrix

```
template<class T>
void OEGeom3DEulerToRotMatrix(T* RMat, const double* angles)
template<class T>
void OEGeom3DEulerToRotMatrix(T* RMat, const double phi,
                              const double theta, const double psi)
```

Constructs a $3x3$ rotation matrix `RMat` from the given Euler angles.

OEGeom3DGetCenterAndExtents

```
template<class T, class U>
void OEGeom3DGetCenterAndExtents(const T* xyz, unsigned int ncoord,
                                 U* center, U* extents)
```

Determines the bounding box that can encompass all the points defined by `xyz` and `ncoord`. The length of each side of the box output in the 3-element array `extents`. The geometric center of the box is output to the `center` parameter.

> **Note**
>
> The center of the box is different from the center-of-mass.

OEGeom3DGetCenterOfMass

```
template<class T, class U>
void OEGeom3DGetCenterOfMass(const T* xyz, unsigned int ncoord, U* com)
template<class T, class U, class Z>
void OEGeom3DGetCenterOfMass(const T* xyz, unsigned int ncoord,
                             U* com, Z* weights)
```

Calculates the center-of-mass for the coordinates given by `xyz` and places the cartesian center-of-mass in the output `com` argument. The number of 3-element cartesian coordinates is specified by the `ncoord` parameter. Note, `weights` can be specified to weight each of the cartesian points differently.

OEGeom3DMagnitude

```
template<class T>
double OEGeom3DMagnitude(T* x)
```

Returns the length of the 3-element vector `x`.

OEGeom3DMakeQuaternion

```
template<class T, class U, class V>
void OEGeom3DMakeQuaternion(T* quat, const U* vec, const V angle)
```

Construct a 4-element quaternion pointed to by `quat` from the 3-element vector `vec` and the angle `angle`. This quaternion represents a 3D rotation around the axis of `vec` by the angle `angle`.

> **Warning**
>
> Translation is not taken into account by this function.

OEGeom3DMatrixIdentity

```
template<class T>
void OEGeom3DMatrixIdentity(T* rmat)
```

Initializes the $3x3$ matrix `rmat` to be an identity matrix: diagonal terms are `1.0`; off-diagonal terms are `0.0`.

OEGeom3DMatrixInvert

```
template<class T, class U>
bool OEGeom3DMatrixInvert(T* mi, const U* m)
```

Calculates the inverse of the matrix `m` and stores the result in `mi`. By definition, multiplying `m` times `mi` will return the identity matrix.

> **Note**
>
> Rotation matrices are orthogonal matrices that can be inverted much more cheaply by using *OEGeom3DMatrixTranspose* instead.

OEGeom3DMatrixMultiply

```
template<class T, class U>
void OEGeom3DMatrixMultiply(T* accumulator, const U* add)
template<class T, class U, class V>
void OEGeom3DMatrixMultiply(T* Product, const U* init, const V* add)
```

Performs $3x3$ matrix multiplication. The first form computes $accumulator* = add$, and the second form computes $Product = init * add$. All of the inputs are nine element $3x3$ matrices.

OEGeom3DMatrixTranspose

```
template<class T>
void OEGeom3DMatrixTranspose(T* mat)
```

Transposes the elements of the $3x3$ matrix `mat`, that is, the elements are swapped along the diagonal.

OEGeom3DNormalize

```
template<class T>
void OEGeom3DNormalize(T* x)
```

Normalizes the 3-element vector `x` so that the length (magnitude) of the vector `x` will be equal to `1.0`.

OEGeom3DNormalizeQuaternion

```
template<class T>
void OEGeom3DNormalizeQuaternion(T* quaternion)
```

Normalizes the four dimensional vector `quaternion` so that it has a magnitude of `1.0`.

OEGeom3DPlanarCoord

```
template<class T>
void OEGeom3DPlanarCoord(T* dst, const T* center, const T* r1,
                         const T* r2, T len)
```

Calculates cartesian coordinate point `dst` that has a distance of `len` from the point `center`, placed on the plane defined by the vectors $(center - r1)$ and $(center - r2)$ and on the line bisecting these two vectors.

OEGeom3DQuaternionIdentity

```
template<class T>
void OEGeom3DQuaternionIdentity(T* quat)
```

Initializes the four element quaternion `quat` to be an identity quaternion: real component is `1.0` and imaginary components are `0.0`.

OEGeom3DQuaternionInvert

```
template<class T, class U>
void OEGeom3DQuaternionInvert(T* iquat, const U* quat)
```

Calculates the inverse `iquat` of the given four element quaternion `quat`. The inverse is also a four element quaternion.

OEGeomQuaternionMultiply

```
template<class T, class U, class V>
void OEGeomQuaternionMultiply(T* Product, const U* qLhs, const V* qRhs)
```

Calculates the product of multiplication of two quaternions, $Product = qLhs * qRhs$.

OEGeom3DQuaternionRotate

```
template<class T, class U>
void OEGeom3DQuaternionRotate(T* xyz, const U* quat,
                              unsigned int ncoord)
template<class T, class U>
void OEGeom3DQuaternionRotate(T* xyz, const U* quat,
                              unsigned int* idx, unsigned int ncoord)
```

Rotates the coordinates of `xyz` using the quaternion given by `quat`. The second form only rotates the coordinates according to the indices given by `idx`.

xyz
: Array of coordinates to rotate

quat
: quaternion for rotation

ncoord
Number of coordinate sets to rotate

idx
Indices of the coordinates to rotate

OEGeom3DQuaternionToRotMatrix

```
template<class T, class U>
void OEGeom3DQuaternionToRotMatrix(T* RMat, const U* quat)
```

Constructs a $3x3$ rotation matrix `RMat` from the given four element quaternion `quat`.

OEGeom3DReflect

```
template<class T>
void OEGeom3DReflect(T* xyz, unsigned int axis_to_reflect_around,
                     unsigned int ncoord)
```

Reflects the coordinates of `xyz` around the axis given by `axis_to_reflect_around`. `axis_to_reflect_around` should be either 0, 1, or 2 for the X, Y, and Z axis respectively.

xyz
Array of coordinates to reflect

axis_to_reflect_around
The axis to reflect around

ncoord
Number of coordinate sets to reflect

OEGeom3DReflectCoord

```
template<class T>
void OEGeom3DReflectCoord(T* dst, const T* center, const T* r1,
                          const T* r2, const T* rHyd)
```

This is a template function that takes T* coordinates of a central atom (`center`) and two r-group atoms (`r1` and `r2`), and performs a reflection around the center-r1-r2 plane for the hydrogen atom (`rHyd`). The output coordinates are returned within the `dst` variable.

OEGeom3DRotVectorToTransform

```
template<class T, class U, class V>
bool OEGeom3DRotVectorToTransform(T* m, T* tran, const U* p1,
                                  const U* p2, const V theta)
```

Return the 3x3 rotation matrix `m` and 3 element translation `tran` for the transformation needed to rotate cartesian coordinates by `theta` around the vector formed by the points `p1` and `p2`.

OEGeom3DRotate

```
template<class T, class U>
void OEGeom3DRotate(T* xyz, const U* m, unsigned int ncoord)
template<class T, class U>
```

(continues on next page)

(continued from previous page)

```
void OEGeom3DRotate(T* xyz, const U* m, unsigned int* idx,
                    unsigned int ncoord)
```

Rotates the coordinates of `xyz` using the $3x3$ rotation matrix given by `m`. The second form only rotates the coordinates according to the indices given by `idx`.

xyz
Array of coordinates to rotate

r
rotation matrix

ncoord
Number of coordinate sets to rotate

idx
Indices of the coordinates to rotate

OEGeom3DScale

```
template<class T>
void OEGeom3DScale(T* x, const T p)
template<class T>
void OEGeom3DScale(T* x, const T p, const T* q)
```

Three dimensional vector scaling. The first form computes $x* = p$ and the second form computes $x = p * q$.

OEGeom3DSetLength

```
template<class T>
void OEGeom3DSetLength(T* x, T len)
```

Sets the length (magnitude) of the three-element vector `x` to the specified value `len`.

OEGeom3DSubtract

```
template<class T, class U>
void OEGeom3DSubtract(T* x, const U* p)
template<class T>
void OEGeom3DSubtract(T* x, const T* p, const T* q)
```

Three dimensional vector subtraction. The first form computes $x- = p$ and the second form computes $x = p - q$.

OEGeom3DTetraCoord

```
template<class T>
void OEGeom3DTetraCoord(T* dst, const T* center, const T* r1,
                        const T* r2, T len)
template<class T>
void OEGeom3DTetraCoord(T* dst, const T* center, const T* r1,
                        const T* r2, const T* r3, T len)
```

Calculates cartesian coordinate point `dst` that has a distance of `len` from the point `center`, and corresponds to a tetrahedral atom location with respect to the given coordinates. The first form places the new coordinates at the known

tetrahedral angle based on the bisecting vector between $(r1 - center)$ and $(r2 - center)$. The second form places it on the direction perpendicular to the plane of the three given coordinates `r1`, `r2`, and `r3`.

OEGeom3DTorsion

```
template<class T>
double OEGeom3DTorsion(const T* p, const T* q, const T* r, const T* s)
```

Determines the torsional angle in three dimensions between cartesian coordinate points `p`, `q`, `r`, and `s`. The result is in radians.

OEGeom3DTranslate

```
template<class T, class U>
void OEGeom3DTranslate(T* xyz, const U* t, unsigned int ncoord)
template<class T, class U>
void OEGeom3DTranslate(T* xyz, const U* t, unsigned int* idx,
                       unsigned int ncoord)
```

Translates the three dimensional coordinates by the vector passed as the argument `t`. The vector `t` must be of size three and contain the Cartesian translation vector. For `ncoord` sets of coordinates the input array `xyz` is expected to have at least a length of $3 * ncoord$. The second form only translates the coordinates according to the indices given by `idx`.

xyz
: Array of coordinates to translate

t
: Translation vector array

ncoord
: Number of coordinate sets to translate

idx
: Index of the coordinates to translate

OEGeom3DUnitQuaternionRotate

```
template<class T, class U>
void OEGeom3DUnitQuaternionRotate(T* xyz, const U* quat,
                                  unsigned int ncoord)
template<class T, class U>
void OEGeom3DUnitQuaternionRotate(T* xyz, const U* quat,
                                  unsigned int* idx,
                                  unsigned int ncoord)
```

Rotates the coordinates of `xyz` using the unit-quaternion given by `quat`. The second form only rotates the coordinates according to the indices given by `idx`. When it is known that the input quaternion is a unit quaternion, this is more efficient than using *OEGeom3DQuaternionRotate*.

xyz
: Array of coordinates to rotate

quat
: unit-quaternion for rotation

ncoord
: Number of coordinate sets to rotate

idx
Indices of the coordinates to rotate

> **Warning**
>
> Calling this functions when the input quaternion is not a unit quaternion could led to unexpected results.

OEGeom3DVolume

```
template<class T>
double OEGeom3DVolume(const T* p, const T* q, const T* r, const T* s)
```

OEGeomDotProd

```
template<class T>
double OEGeomDotProd(const T* p, const T* q, unsigned int n)
```

Returns the dot-product between the two `n` dimensional vectors `p` and `q`.

OEGeomNormalizeQuaternion

```
template<class T>
void OEGeomNormalizeQuaternion(T* quaternion)
```

OEGeomQuaternionMultiply

```
template<class T, class U, class V>
void OEGeomQuaternionMultiply(T* Product, const U* qLhs, const V* qRhs)
```

OEInBox

```
bool OEInBox(const OEBox& box, const float x, const float y,
             const float z)
```

Returns true if (x , y, z) is within *box*.

Coordinates that lie on an edge of *box* are considered to be in *box*.

OEInSphere

```
bool OEInSphere(const OESphere& sph, const float x, const float y,
                const float z)
```

Returns true if (x , y, z) is within the sphere *sph*.

Coordinates that lie on an edge of *sph* are considered to be in *sph*.

OESetupBox

```
bool OESetupBox(OEBox& box, const float* xyz, const unsigned N,
                const bool addbox = 0.0f)
```

Sets up *box* to contain the minimum size box that encloses all coordinates in the *xyz* array and then extends each box face by *addbox*.

box
Box to setup

xyz
Array of coordinates

N
Number of coordinates in *xyz* array

addbox
Optional parameter to extend each box face by a specified amount relative to the minimum box enclosing the *xyz* coordinates.

Returns true if setup was successful.

OESetupBoxCenterAndExtents

```
bool OESetupBoxCenterAndExtents(OEBox& box, const float* center,
                                const float* extents)
```

Sets up *box* using a center coordinate and the dimension of the box.

box
Box to setup

center
A length 3 array holds the coordinate of the box center

extends
A length 3 array holding the x, y and z dimensions of the box.

Returns true if setup was successful.

OESphereArea

```
float OESphereArea(const OESphere& sph)
```

Calculates the area of the sphere

OESphereVolume

```
float OESphereVolume(const OESphere& sph)
```

Calculate the volume of the sphere.

4.7 OEBio API

4.7.1 OEBio Classes

OEAltGroup

```
class OEAltGroup
```

This class represents *OEAltGroup*, an abstraction of a particular alternate location group in a molecule managed by an *OEAltLocationFactory*. It can be thought of as a collection of *OEAltLocation* objects, each referring to a specific set of atomic positions.

The following *OEAltLocationFactory* methods return an *OEAltGroup* or the corresponding iterator:

GetGroup	*GetGroups*	*GetGroup*

The following *OEAltLocationFactory* methods take an *OEAltGroup*:

GetLocationCodes	*GetAltAtoms*	*GetCurrentLocations*

Constructors

```
OEAltGroup()
OEAltGroup(const OEAltGroup& grp)
OEAltGroup(const unsigned int* counts, unsigned int numGrps)
```

Default, copy and initialization constructors.

If no arguments are supplied, the *OEAltGroup* is uninitialized and will return `false` when *operator bool* is called.

If an *OEAltGroup* is supplied, the constructed object is identical to the input.

If an array of location counts is supplied along with the total number of groups (the size of the array), an object is constructed that represents the first of these groups (and can represent any of these groups).

operator bool

```
operator bool() const
```

Returns `true` if the object refers to an actual group. Returns `false` if the group identifier is out of range.

GetGroupCount

```
unsigned int GetGroupCount() const
```

Returns the total number of alternate location groups found by the corresponding *OEAltLocationFactory*.

GetGroupID

```
int GetGroupID() const
```

Returns an integer that identifies this alternate location group. A negative number indicates the object does not refer to an actual alternate location group.

GetLocationCount

```
unsigned int GetLocationCount() const
```

Returns the total number of *OEAltLocations* in this group.

GetLocations

```
OESystem::OEIterBase<const OEAltLocation> *GetLocations() const
```

Returns an iterator over all *OEAltLocations* in this group.

SetGroupID

```
OEAltGroup& SetGroupID(int newGrp)
```

Changes the group identifier for this group. If the identifier is out of range, *operator bool* will return `false`.

OEAltLocation

```
class OEAltLocation
```

This class represents *OEAltLocation*, an abstraction of a particular alternate location of an *OEAltGroup* in the molecule used to construct an *OEAltLocationFactory*.

The following *OEAltLocationFactory* methods return an *OEAltLocation* or the corresponding iterator:

GetLocation	*GetCurrentLocations*

The following *OEAltGroup* method returns an *OEAltLocation* iterator:

GetLocations

The following *OEAltLocationFactory* methods take an *OEAltLocation*:

GetLocationCodes	*SetAlt*	*MakeAltMol*	*GetGroup*

Constructors

```
OEAltLocation()
OEAltLocation(const OEAltLocation& loc)
OEAltLocation(int grpID, int locID, unsigned int numLocs)
```

Default, copy and initialization constructors.

If no arguments are supplied, the *OEAltLocation* is uninitialized and will return `false` when *operator bool* is called.

If an *OEAltLocation* is supplied, the constructed object is identical to the input.

If a group ID, a location ID and the total number of locations in the group is supplied, an object is constructed that represents the specified location in the specified group.

operator bool

```
operator bool() const
```

Returns `true` if the object refers to an actual location. Returns `false` if the location identifier is out of range.

GetGroupID

```
int GetGroupID() const
```

Returns an integer that identifies the *OEAltGroup* that contains this location. A negative number indicates the object does not refer to an actual alternate location group.

GetLocationCount

```
unsigned int GetLocationCount() const
```

Returns the total number of alternate locations in the *OEAltGroup* that contains this location.

GetLocationID

```
int GetLocationID() const
```

Returns an integer that identifies this location. A negative number indicates the object does not refer to an actual alternate location.

SetLocationID

```
OEAltLocation& SetLocationID(int newLoc)
```

Changes the location identifier for this location. If the identifier is out of range, *operator bool* will return `false`.

OEAltLocationFactory

```
class OEAltLocationFactory
```

This class represents *OEAltLocationFactory*, a class to perceive and manage alternate locations in a molecule and generate new subset molecules for specific sets of *OEAltLocation* selections.

Constructors

```
OEAltLocationFactory(const OEAltLocationFactory& fact)
OEAltLocationFactory(const OEChem::OEMolBase& src,
                     unsigned int options = OEBio::OEAltLocFactOption::None)
```

Default and copy constructors.

If an *OEMolBase* is supplied, the constructed factory will describe alternate locations in the input molecule and generate new single-conformation molecules on demand for any set of alternate location selections. The initial selection state of this factory is as if *SetPrimaryAlt* has been called.

> **Note**
>
> When reading a molecule from a PDB file that will be used to construct an *OEAltLocationFactory* be sure to specify the input flavor: *OEIFlavor.PDB.ALTLOC* so that all the alternate location atoms will be retained.

The options argument provides control over how the factory is constructed. The set of unsigned values for this parameter is specified by the *OEAltLocFactOption* namespace.

If an *OEAltLocationFactory* is supplied, an object is constructed that matches the input.

GetAltAtoms

```
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAltAtoms() const
OESystem::OEIterBase< OEChem::OEAtomBase> * GetAltAtoms()
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAltAtoms(const␣
↪OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred) const
OESystem::OEIterBase<OEChem::OEAtomBase>* GetAltAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase>& pred)
OESystem::OEIterBase<const OEChem::OEAtomBase> * GetAltAtoms(const OEAltGroup& grp) const
OESystem::OEIterBase< OEChem::OEAtomBase> * GetAltAtoms(const OEAltGroup& grp)
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAltAtoms(const OEAltGroup& grp,
                                                            const␣
↪OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred) const
OESystem::OEIterBase< OEChem::OEAtomBase> * GetAltAtoms(const OEAltGroup& grp,
                                                        const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase>& pred)
OESystem::OEIterBase<const OEChem::OEAtomBase> * GetAltAtoms(const OEAltLocation& loc)␣
↪const
OESystem::OEIterBase< OEChem::OEAtomBase> * GetAltAtoms(const OEAltLocation& loc)
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAltAtoms(const OEAltLocation& loc,
                                                            const␣
↪OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred) const
OESystem::OEIterBase< OEChem::OEAtomBase> * GetAltAtoms(const OEAltLocation& loc,
                                                        const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase>& pred)
```

Returns an *OEAtomBase* iterator over the atoms with alternate locations. Both `const` and non-`const` versions are supplied for each iterator.

If no arguments are supplied, the iterator is over all alternate location atoms.

If an *OEAltGroup* or *OEAltLocation* is supplied, the iterator is only over the atoms in the specified group or location.

If an *OEUnaryPredicate* is supplied, the iterator will return those alternate location atoms that also match the predicate.

GetAtomMapSize

```
unsigned int GetAtomMapSize() const
```

Returns the minimum size of the array of *OEAtomBase* pointers the user must supply to one of the *MakePrimaryAltMol*, *MakeCurrentAltMol*, or *MakeAltMol* methods if an atom map is to be returned. Atom maps are pointers to atoms in the generated molecule at locations in the array equal to the internal index of atoms in the molecule returned by *GetSourceMol*, or 0 for atoms not included in the generated molecule.

GetBondMapSize

```
unsigned int GetBondMapSize() const
```

Returns the minimum size of the array of *OEBondBase* pointers the user must supply to one of the *MakePrimaryAltMol*, *MakeCurrentAltMol*, or *MakeAltMol* methods if a bond map is to be returned. Bond maps are arrays of pointers to bonds in the generated molecule at locations in the array equal to the internal index of bonds in the molecule returned by *GetSourceMol*, or 0 for bonds not included in the generated molecule.

GetCurrentAtoms

```
OESystem::OEIterBase< OEChem::OEAtomBase> * GetCurrentAtoms()
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetCurrentAtoms() const
OESystem::OEIterBase< OEChem::OEAtomBase> * GetCurrentAtoms(const␣
↪OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred)
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetCurrentAtoms(const␣
↪OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred) const
```

Returns an *OEAtomBase* iterator over the atoms in the current selection – atoms in selected locations in each group plus atoms that are not in any alternate location group. Both `const` and non-`const` versions are supplied for each iterator.

If an *OEUnaryPredicate* is supplied, the iterator will return those selected atoms that also match the predicate.

GetCurrentLocations

```
OESystem::OEIterBase<const OEAltLocation>* GetCurrentLocations() const
OESystem::OEIterBase<const OEAltLocation>* GetCurrentLocations(const OEAltGroup& grp)␣
↪const
OESystem::OEIterBase<const OEAltLocation>* GetCurrentLocations(int grpNum) const
```

Returns an iterator over currently selected *OEAltLocations*. If no argument is supplied, every selected location is returned, otherwise only the selected location(s) for the specified *OEAltGroup* is returned. Unless *SetAlt* has been called with `multiLoc` set to `true`, only a single *OEAltLocation* will be returned per *OEAltGroup*.

GetGroup

```
OEAltGroup GetGroup(const OEChem::OEAtomBase* selAtom) const
OEAltGroup GetGroup(const OEAltLocation& loc) const
```

If an *OEAtomBase* pointer is supplied, the method returns the *OEAltGroup* that includes the residue of the atom provided as input. If the input atom residue is not in any of the alternate location groups, a warning will be thrown, the *OEAltGroup* will be uninitialized and will return `false` when *operator bool* is called.

If an *OEAltLocation* is supplied, the method returns the *OEAltGroup* the location is associated with.

GetGroupCount

```
unsigned int GetGroupCount() const
```

Returns the total number of *OEAltGroups* identified by this *OEAltLocationFactory*.

GetGroups

```
OESystem::OEIterBase<const OEAltGroup> *GetGroups() const
```

Returns an iterator over all *OEAltGroups* managed by this *OEAltLocationFactory*.

GetLocation

```
OEAltLocation GetLocation(const OEChem::OEAtomBase* selAtom,
                          char confCode, bool caseSensitive = true) const
```

Returns the *OEAltLocation* that includes the residue of the selected atom and the conformation code provided as input. The user can select whether or not the conformation code match is case sensitive. If the input atom residue is not in any of the alternate location groups, a warning will be thrown, the *OEAltLocation* will be uninitialized and will return `false` when *operator bool* is called.

GetLocationCodes

```
std::string GetLocationCodes(const OEAltGroup& grp) const
std::string GetLocationCodes(const OEAltLocation& loc) const
```

Returns a string listing all the alternate location characters in the input *OEAltGroup* or *OEAltLocation*. Although most *OEAltLocations* will return a string containing only a single character, some that are more complicated will return a string with more than one character.

GetSourceMol

```
const OEChem::OEMolBase& GetSourceMol() const
```

Returns a constant reference to a copy of the *OEMolBase* supplied to the *Constructor*.

GetState

```
OEAltLocationState GetState() const
void GetState(OEAltLocationState& state) const
```

Returns an *OEAltLocationState* that represents the current selection state of this *OEAltLocationFactory*.

See also

- *SetState*
- *MakeAltMol*

MakeAltMol

```
bool MakeAltMol(OEChem::OEMolBase& dst, const OEAltLocation& loc,
                OEChem::OEAtomBase** atomMap = 0,
                OEChem::OEBondBase** bondMap = 0,
                unsigned int options = OEBio::OEMakeAltMolOption::None)
bool MakeAltMol(OEChem::OEMolBase& dst,
                const OEChem::OEAtomBase* selAtom, char confCode,
                bool caseSensitive = true,
                OEChem::OEAtomBase** atomMap = 0,
                OEChem::OEBondBase** bondMap = 0,
                unsigned int options = OEBio::OEMakeAltMolOption::None)
bool MakeAltMol(OEChem::OEMolBase& dst,
                const OEAltLocationState& state,
                OEChem::OEAtomBase** atomMap = 0,
                OEChem::OEBondBase** bondMap = 0,
                unsigned int options = OEBio::OEMakeAltMolOption::None)
```

Sets the selection state of this *OEAltLocationFactory* and generates a new subset molecule (`dst`) for this state, returning a value indicating whether the selection was valid. Unless *SetAlt* has been called with `multiLoc` set to `true`, the subset mol will represent a single conformation.

If either a location or an atom selection is supplied (see *GetLocation*), the selection state of only one group is set. If an *OEAltLocationState* is supplied, the selection state of all groups may be affected (see *GetState*).

If either a location or an atom selection is supplied and cannot be found in any alternation group, the `dst` molecule is not updated and `false` is returned.

If an array of *OEAtomBase* pointers is provided, this method will fill it with an atom map: pointers to atoms in the generated molecule at locations in the array equal to the internal index of atoms in the molecule returned by *GetSourceMol*, or 0 for atoms not included in the generated molecule.

If an array of *OEBondBase* pointers is provided, this method will fill it with a bond map: pointers to bonds in the generated molecule at locations in the array equal to the internal index of bonds in the molecule returned by *GetSourceMol*, or 0 for bonds not included in the generated molecule.

The options argument provides control over how the molecule is constructed. The set of unsigned values for this parameter is specified by the *OEMakeAltMolOption* namespace.

MakeCurrentAltMol

```
void MakeCurrentAltMol(OEChem::OEMolBase& dst,
                       OEChem::OEAtomBase** atomMap = 0,
                       OEChem::OEBondBase** bondMap = 0,
                       unsigned int options = OEBio::OEMakeAltMolOption::None) const
```

Generates a new subset molecule (`dst`) for the current selection state. Unless *SetAlt* has been called with `multiLoc` set to `true`, the subset mol will represent a single conformation.

If an array of *OEAtomBase* pointers is provided, this method will fill it with an atom map: pointers to atoms in the generated molecule at locations in the array of the internal index of atoms in the molecule returned by *GetSourceMol*, or 0 for atoms not included in the generated molecule.

If an array of *OEBondBase* pointers is provided, this method will fill it with a bond map: pointers to bonds in the generated molecule at locations in the array of the internal index of bonds in the molecule returned by *GetSourceMol*, or 0 for bonds not included in the generated molecule.

The options argument provides control over how the molecule is constructed. The set of unsigned values for this parameter is specified by the *OEMakeAltMolOption* namespace.

MakePrimaryAltMol

```
void MakePrimaryAltMol(OEChem::OEMolBase& dst,
                       OEChem::OEAtomBase** atomMap = 0,
                       OEChem::OEBondBase** bondMap = 0,
                       unsigned int options = OEBio::OEMakeAltMolOption::None)
```

Resets the selection state of this *OEAltLocationFactory* to the primary location (see *SetPrimaryAlt*) and generates a new subset molecule (`dst`) for this state. Unless *SetAlt* has been called with `multiLoc` set to `true`, the subset mol will represent a single conformation.

If an array of *OEAtomBase* pointers is provided, this method will fill it with an atom map: pointers to atoms in the generated molecule at locations in the array of the internal index of atoms in the molecule returned by *GetSourceMol*, or 0 for atoms not included in the generated molecule.

If an array of *OEBondBase* pointers is provided, this method will fill it with a bond map: pointers to bonds in the generated molecule at locations in the array of the internal index of bonds in the molecule returned by *GetSourceMol*, or 0 for bonds not included in the generated molecule.

The options argument provides control over how the molecule is constructed. The set of unsigned values for this parameter is specified by the *OEMakeAltMolOption* namespace.

SetAlt

```
bool SetAlt(const OEAltLocation& loc, bool multiLoc = false)
bool SetAlt(const OEChem::OEAtomBase* selAtom, char confCode,
            bool caseSensitive = true)
```

If an *OEAltLocation* is supplied, the selection state of the *OEAltGroup* containing the location is set to select that location, with a return value indicating whether the location was found. If the optional `multiLoc` parameter has the (default) value of `false`, the function acts like a toggle: the previous location for this group is unselected before the new location is selected.

For a description of the atom selection form, see *GetLocation*.

Warning

If this method is called with the `multiLoc` parameter set to `true`, the *OEAltLocationFactory* may be left in a state where *MakeCurrentAltMol* no longer generates a single-conformation molecule.

SetPrimaryAlt

```
void SetPrimaryAlt()
```

Resets the selection state of this *OEAltLocationFactory* to the first location for each group – the location with the highest average occupancy.

SetState

```
bool SetState(const OEAltLocationState& state)
```

Resets the selection state of this *OEAltLocationFactory* to that represented by the input *OEAltLocationState*, with a return value indicating whether the state was valid.

See also

- *GetState*
- *MakeAltMol*

OEAltLocationState

```
class OEAltLocationState
```

This class represents *OEAltLocationState*, an abstraction of the internal *OEAltLocation* selection state of an *OEAltLocationFactory*. It is designed for ease of use with collection classes and to support serialization.

The following *OEAltLocationFactory* methods get or set an *OEAltLocationState*:

GetState	*SetState*	*MakeAltMol*

Constructors

```
OEAltLocationState()
OEAltLocationState(const std::string& hex)
OEAltLocationState(const OEAltLocationState& state)
```

Default and copy constructors.

If no arguments are supplied, the constructed object is uninitialized and will return `false` when *operator bool* is called.

If a `string` is supplied, then an object is constructed to represent the state described by the string. Strings appropriate for use with this method are generated by *OEAltLocationState.ToHexString*. This constructor is used when reconstituting the serialized state of an *OEAltLocationFactory* prior to providing the resulting *OEAltLocationState* to *OEAltLocationFactory.MakeAltMol*.

If an *OEAltLocationState* is supplied, an object is constructed that is identical to the input.

operator bool

```
operator bool() const
```

Returns `true` if the state has been initialized (constructed from a `string` argument or from another state that has been initialized). Otherwise, returns `false`.

ToHexString

```
bool ToHexString(std::string& hex) const
```

Generate a text string that describes the state. Used when serializing the state of an *OEAltLocationFactory*.

OEAtomMatchResidue

```
class OEAtomMatchResidue : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEAtomMatchResidue* functor that identifies atoms (*OEAtomBase*) in residues (*OEResidue*) matching specific pieces, like residue name, residue number, insertion code, chain ID, and fragment number.

See also

- *Residue Data Functors* section
- *OEAtomMatchResidueID* class

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEAtomMatchResidue(const std::string& resId)
OEAtomMatchResidue(const std::vector<std::string>& resIds)
OEAtomMatchResidue(const OEAtomMatchResidueID& resId)
OEAtomMatchResidue(const std::vector<OEAtomMatchResidueID>& resIds)
OEAtomMatchResidue(const OEChem::OEResidue& resId)
OEAtomMatchResidue(const std::vector<OEChem::OEResidue>& resIds)
```

Constructs the functor with the residue properties specified by a regex string, an *OEAtomMatchResidueID* or a vector of those, an OEResidue, or a vector of those.

```
OEAtomMatchResidue(const OEChem::OEMolBase& molA,
                   const OEChem::OEMolBase& molB,
                   const double dist, bool excludeHydrogens = true)
```

Constructs the functor to match the complete residues in *molA* that are within the specific distance *dist* in *molB*. An option exists to exclude explicit hydrogens to be considered inside the distance threshold, which is on by default.

operator()

```
bool operator()(const OEAtomBase& atom) const
```

Returns `true`, if the residue of the atom (returned by the *OEAtomGetResidue* function) has equivalent residue properties (i.e. residue name, residue number, insertion code, chain ID, and fragment number) with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEAtomMatchResidue* object is dynamically allocated and owned by the caller.

Examples

The examples below demonstrate how to create various predicates that allow to access atoms based on their residue information.

OEAtomMatchResidueID

```
class OEAtomMatchResidueID
```

The *OEAtomMatchResidueID* class stores residue properties, such as residue name, residue number, insertion code, chain ID, fragment number, and alternate location. It is used to construct an *OEAtomMatchResidue* predicate that provides a convenient way to iterate over atoms with specific residue information.

See also

- *OEResidue* class
- *Residue Data Functors* section
- *OEAtomMatchResidue* predicate

Code Example

- See code examples in the *OEAtomMatchResidue* predicate

Constructors

```
OEAtomMatchResidueID()
```

Default constructor that initializes the object with fuzzy regex matches `.*` for each property.

```
OEAtomMatchResidueID(OEChem::OEResidue res)
```

Constructs an *OEAtomMatchResidueID* object with the given residue.

Note

There is a difference between using an OEResidue object retrieved from an existing atom, and a default constructed OEResidue object where the user has set a few variables. This is because the default constucted OEResidue object sets the residue name to `UNL`, residue number to `1`, fragment number to `0`, chainID, insert code, and alternate location to blank characters `` ``. Because of these defaults, when constructing from an OEResidue object, we convert fragment number `` ``0``, to a fuzzy regex match `.*`, since it will be 1 or higher if it was read from a PDB file. In the same manner a blank `` `` `` character for the chainID is converted into a fuzzy regex match. However, since a blank character is allowed and the most common for an insertCode and alternate location, these are kept as is and not converted.

GetName

```
std::string GetName() const
```

By default, it returns `.*` to match atoms with any residue name.

GetResidueNumber

```
std::string GetResidueNumber() const
```

By default, it returns `.*` to match atoms with any residue number.

GetInsertCode

```
std::string GetInsertCode() const
```

By default, it returns `.*` to match atoms with any residue insert code.

GetChainID

```
std::string GetChainID() const
```

By default, it returns `.*` to match atoms with any residue chain ID.

GetFragmentNumber

```
std::string GetFragmentNumber() const
```

By default, it returns `.*` to match atoms with any residue fragment number.

GetAlternateLocation()

```
std::string GetAlternateLocation() const
```

By default, it returns `.*` to match atoms with any alternate location code.

SetName

```
void SetName(const std::string& n)
```

See also *OEResidue.SetName* method.

SetResidueNumber

```
void SetResidueNumber(const std::string& n)
```

See also *OEResidue.SetResidueNumber* method.

SetInsertCode

```
void SetInsertCode(const std::string& i)
```

See also *OEResidue.SetInsertCode* method.

SetChainID

```
void SetChainID(const std::string& c)
```

See also *OEResidue.SetChainID* method.

SetFragmentNumber

```
void SetFragmentNumber(const std::string& f)
```

See also *OEResidue.SetFragmentNumber* method.

SetAlternateLocation

```
void SetAlternateLocation(const std::string& f)
```

See also *OEResidue.SetAlternateLocation* method.

OECationPiInteractionHint

```
class OECationPiInteractionHint : public OEInteractionHintTypeBase
```

The *OECationPiInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as **cation-Pi** interactions (inter- and intramolecular).

See also

- *OECationPiInteractionHintType* namespace for specific **cation-Pi** interaction types
- *OEIsCationPiInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **cation-Pi** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its methods:
  - *OEPerceiveInteractionOptions.SetMaxCationPiAngle*
  - *OEPerceiveInteractionOptions.SetMaxCationPiDistance*

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OECationPiInteractionHint(unsigned int type)
```

Creates an *OECationPiInteractionHint* object with the given type.

type
This value has to be from the *OECationPiInteractionHintType* namespace.

Note

Two *OECationPiInteractionHint* objects with different types from the *OECationPiInteractionHintType* namespace are considered to be different.

```
OECationPiInteractionHint(const OECationPiInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OECationPiInteractionHint* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **cation-Pi** interaction when added to an *OEInteractionHintContainer* object.

It checks that each fragment contains at least one atom, and the fragments belong to different molecules *i.e.* it is an **intermolecular** interaction.

OEChelatorInteractionHint

```
class OEChelatorInteractionHint : public OEInteractionHintTypeBase
```

The *OEChelatorInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as **chelator (metal)** interactions (inter- and intramolecular).

See also

- *OEChelatorInteractionHintType* namespace for specific **chelator** interaction types
- *OEIsChelatorInteractionHint* predicate
- *OEIsIntramolecularChelatorInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **chelator** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its *OEPerceiveInteractionOptions.SetMaxMetalBondDistance* method

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OEChelatorInteractionHint(unsigned int type)
```

Constructor to create a **chelator** interaction type with the given parameter.

type
This value has to be from the *OEChelatorInteractionHintType* namespace.

> **Note**
>
> Two *OEChelatorInteractionHint* objects with different types from the *OEChelatorInteractionHintType* namespace are considered to be different.

```
OEChelatorInteractionHint(const OEChelatorInteractionHint& rhs)
```

Copy Constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEChelatorInteractionHint* object is dynamically allocated and owned by the caller.

GetChelatorType

```
unsigned int GetChelatorType() const
```

Returns the type of the *OEChelatorInteractionHint* object. The return value is taken from the *OEChelatorInteractionHintType* namespace.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **chelator** interaction when added to an *OEInteractionHintContainer* object.

It checks that each fragment contains exactly one atom, and the fragments belong to different molecules *i.e.* it is an **intermolecular** interaction.

OEClashInteractionHint

```
class OEClashInteractionHint : public OEInteractionHintTypeBase
```

The *OEClashInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as intermolecular **clash** interactions.

See also

- *OEIsClashInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **clash** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its *OEPerceiveInteractionOptions.SetMinContactFraction* method

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OEClashInteractionHint()
```

Default constructor.

```
OEClashInteractionHint(const OEClashInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEClashInteractionHint* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **clash** interaction when added to an *OEInteractionHintContainer* object.

It checks that each fragment contains exactly one atom, and the fragments belong to different molecules *i.e.* it is an **intermolecular** interaction.

OEContactInteractionHint

```
class OEContactInteractionHint : public OEInteractionHintTypeBase
```

The *OEContactInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as intermolecular **contact** interactions.

See also

- *OEIsContactInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **contact** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its methods:
  - *OEPerceiveInteractionOptions.SetMinContactFraction*
  - *OEPerceiveInteractionOptions.SetMaxContactFraction*

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OEContactInteractionHint()
```

Default constructor.

```
OEContactInteractionHint(const OEContactInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEContactInteractionHint* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **contact** interaction when added to an *OEInteractionHintContainer* object.

It checks that each fragment contains exactly one atom, and the fragments belong to different molecules *i.e.* it is an **intermolecular** interaction.

OECovalentInteractionHint

```
class  OECovalentInteractionHint : public OEInteractionHintTypeBase
```

The *OECovalentInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as intermolecular **covalent** interactions.

See also

- *OEIsCovalentInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **covalent** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its methods:
  - *OEPerceiveInteractionOptions.SetMaxCovalentFraction*
  - *OEPerceiveInteractionOptions.SetMinCovalentFraction*

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OECovalentInteractionHint()
```

Default constructor.

```
OECovalentInteractionHint(const OECovalentInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OECovalentInteractionHint* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **covalent** interaction when added to an *OEInteractionHintContainer* object.

It checks that each fragment contains exactly one atom, and the fragments belong to different molecules *i.e.* it is an **intermolecular** interaction.

OECrystalSymmetryParams

```
class OECrystalSymmetryParams
```

This class represents the *OECrystalSymmetryParams* class that stores crystal symmetry parameters:

- standard cell lengths
- cell angles 'alpha', 'beta' and 'gamma' and
- spacegroup number
- z-value

See also

- *OEGetCrystalSymmetry* function
- *OESetCrystalSymmetry* function

Constructors

```
OECrystalSymmetryParams()
```

Default constructor.

```
OECrystalSymmetryParams(float a, float b, float c, float alpha,
                        float beta, float gamma,
                        unsigned int sgnumber,
                        unsigned int zvalue = 0u)
```

Constructors an *OECrystalSymmetryParams* object with the given values.

GetA

```
float GetA() const
```

GetAlpha

```
float GetAlpha() const
```

GetB

```
float GetB() const
```

GetBeta

```
float GetBeta() const
```

GetC

```
float GetC() const
```

GetGamma

```
float GetGamma() const
```

GetSpaceGroup

```
unsigned int GetSpaceGroup() const
```

GetZValue

```
unsigned int GetZValue() const
```

SetA

```
void SetA(float a)
```

SetAlpha

```
void SetAlpha(float alpha)
```

SetB

```
void SetB(float b)
```

SetBeta

```
void SetBeta(float beta)
```

SetC

```
void SetC(float c)
```

SetGamma

```
void SetGamma(float gamma)
```

SetSpaceGroup

```
void SetSpaceGroup(unsigned int sgnumber)
```

SetZValue

```
void SetZValue(unsigned int zvalue)
```

OEDesignUnit

Attention

This API is currently available in C++ and Python.

```
class OEDesignUnit : public OESystem::OEBase
```

This is a container class derived from *OEBase* that holds all molecules needed to construct a model-ready biomolecular system.

Constructors

Molecule constructors

Note

Molecule constructors are intended for already prepared molecules. They requires a target structure, as well as either a ligand or a list of binding site residues. Solvent and co-factors are optional, and are only set if the molecules passed are valid. The target molecule is not split into components and no structure preparation take place. A title will be generated based on the names of the provided molecules, but can be set manually afterwards.

```
OEDesignUnit(const OEChem::OEMolBase& target,
             const OEChem::OEMolBase& ligand,
             const OEChem::OEMolBase& solvent = OEChem::OEGraphMol(),
             const OEChem::OEMolBase& cofactors = OEChem::OEGraphMol(),
             const unsigned targetID = OEDesignUnitComponents::Protein)
```

Target-ligand constructor

```
OEDesignUnit(const OEChem::OEMolBase& target,
             const std::vector<std::string>& siteResidues,
             const OEChem::OEMolBase& solvent = OEChem::OEGraphMol(),
             const OEChem::OEMolBase& cofactors = OEChem::OEGraphMol(),
             const unsigned targetID = OEDesignUnitComponents::Protein)
```

Apo target constructor

operator=

Assignment operator.

operator bool

Clear

CreateCopy

GetComponent

GetComponentID

GetComponentTag

GetComponents

GetComponentsWithin

GetDataType

GetIDs

GetLigand

GetMetal

GetNucleicAcid

GetPDBMetaData

GetPackingResidues

GetProtein

GetReceptor

GetSiteResidues

GetSolvent

GetStructureQuality

See also

- *OEStructureQuality* class

GetTransform

See also

- *OETrans* class

GetTaggedComponents

GetTags

GetTarget

GetTargetComponentID

GetTargetComponentTag

GetTitle

GetAltLoc

HasAltLoc

HasComponent

HasLigand

HasMetal

HasNucleicAcid

HasPDBMetaData

HasPackingResidues

HasProtein

HasReceptor

HasSiteResidues

HasSolvent

HasStructureQuality

HasTransform

HasTarget

HasTitle

IsDataType

SetTitle

SetSiteResidues

SetSiteResidues requires residues that are present in the current target component. SetSiteResidues returns `False` if `siteResidues` is empty, or if any residue in `siteResidues` is not found in the target.

SetSiteResidues uses the optional parameters as follows:

- The `threshold` option defines the minimum heavy-atom fraction of the current ligand that must fall inside the site box. Setting the value to 0.0 skips the ligand coverage validation.
- The `addBox` option expands the site box before ligand coverage is evaluated.

- The `rearrangeComponents` option controls whether components are re-arranged in the design unit based on whether the ligand coverage validation is true or false. Setting `rearrangeComponents` to `False` causes SetSiteResidues to return `False` when ligand coverage is below `threshold`. Setting the value to `True`, allows component reassignment when ligand coverage is below `threshold`. The current ligand is moved to `OtherLigands` and cofactors are moved to `OtherCofactors`. The list of `OtherLigands` is then evaluated, and the best ligand that meets `threshold` can be promoted into the ligand component, with corresponding `OtherCofactors` moved to the `Cofactors` component.
- `siteSize` controls how site-associated cofactors are determined.

Note

SetSiteResidues preserves an existing receptor only when at least one new site residue overlaps the previous site-residue set. SetSiteResidues clears the receptor when no overlap is found.

ClearReceptor

ClearComponent

OEHalogenBondInteractionHint

```
class OEHalogenBondInteractionHint : public OEInteractionHintTypeBase
```

The *OEHalogenBondInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as **halogen bond** interactions (inter- and intramolecular).

See also

- *OEHalogenBondInteractionHintType* namespace for specific **halogen-bond** interaction types
- *OEIsHalogenBondInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **halogen bond** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its methods:
  - *OEPerceiveInteractionOptions.SetMaxHalogenAngle*
  - *OEPerceiveInteractionOptions.SetMaxHalogenBondDistance*

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OEHalogenBondInteractionHint(unsigned int type)
```

type
This value has to be from the *OEHalogenBondInteractionHintType* namespace.

> **Note**
>
> Two *OEHalogenBondInteractionHint* objects with different types from the *OEHalogenBondInteractionHintType* namespace and/or with different charge options are considered to be different.

```
OEHalogenBondInteractionHint(const OEHalogenBondInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHalogenBondInteractionHint* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **halogen bond** interaction when added to an *OEInteractionHintContainer* object.

It checks that each fragment contains exactly one atom, and the fragments belong to different molecules *i.e.* it is an **intermolecular** interaction.

OEHasAlternateLocation

```
class OEHasAlternateLocation : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

> **See also**
>
> *Residue Data Functors* section

This class represents *OEHasAlternateLocation*, a functor that identifies atoms (*OEAtomBase*) with an alternate location code.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator() *CreateCopy*

Constructors

```
OEHasAlternateLocation(const OEHasAlternateLocation& rhs)
OEHasAlternateLocation(char confCode = '\0',
                       bool acceptBlanks = false,
                       bool caseSensitive = true)
```

Default and copy constructors.

If a non-blank `char` is supplied, atoms with that alternate location code will be selected when the object is used in a predicate expression. If a non-blank `char` is not supplied, atoms with any non-blank alternate location code will be selected.

If `acceptBlanks` is `true`, then atoms with blank alternate location codes will also be selected. If `caseSensitive` is `false`, then both upper- and lower-case versions of the `confCode` parameter will be tried in the match.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

Returns `true`, if the *OEAtomBase* has a *OEResidue* and *OEResidue.GetAlternateLocation* returns a `char` that matches the values used in the *OEHasAlternateLocation.Constructors*.

CreateCopy

```
OESystem::OEUnaryPredicate<OEChem::OEAtomBase>::base_type* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasAlternateLocation* object is dynamically allocated and owned by the caller.

OEHasInteractionHint

```
class OEHasInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEHasInteractionHint* functor that identifies interactions (*OEInteractionHint*) with a specific atom (*OEAtomBase*).

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Accessing Interaction Hint Information **OpenEye Python Cookbook** recipe

Constructors

```
OEHasInteractionHint(const OEChem::OEAtomBase* atom)
```

Constructs the functor with the given atom.

```
OEHasInteractionHint(const OEHasInteractionHint& rhs)
```

Copy constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the atom with which the functor is constructed is stored in the given interaction.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasInteractionHint* object is dynamically allocated and owned by the caller.

OEHasInteractionHintType

```
class OEHasInteractionHintType : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEHasInteractionHintType* functor that identifies connections (*OEInteractionHint*) with a specific type (*OEInteractionHintTypeBase*).

See also

- *OEInteractionHintContainer.GetInteractions* method

Constructors

```
OEHasInteractionHintType(const OEInteractionHintTypeBase& itype)
```

Constructs the functor with the given interaction type.

See also

The following built-in interaction hint types are available:

- *OECationPiInteractionHint*
- *OEChelatorInteractionHint*
- *OEClashInteractionHint*
- *OEContactInteractionHint*
- *OECovalentInteractionHint*
- *OEHalogenBondInteractionHint*
- *OEHBondInteractionHint*
- *OESaltBridgeInteractionHint*
- *OEStackingInteractionHint*

```
OEHasInteractionHintType(const OEHasInteractionHintType& rhs)
```

Copy constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the interaction has the same type with which the functor is constructed.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasInteractionHintType* object is dynamically allocated and owned by the caller.

OEHasPDBAtomIndex

```
class OEHasPDBAtomIndex : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEHasPDBAtomIndex* functor that identifies atoms (*OEAtomBase*) with a specific PDB index from the *OEPDBAtomName* namespace.

See also

- *Residue Data Functors* section
- *OEGetPDBAtomIndex* function

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasPDBAtomIndex(unsigned pdbatomidx)
```

Default and copy constructors.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasPDBAtomIndex* object is dynamically allocated and owned by the caller.

OEHasResidueInteractionHint

```
class OEHasResidueInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEHasResidueInteractionHint* functor that identifies connections (*OEInteractionHint*) with a specific residue (*OEResidue*).

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Accessing Interaction Hint Information **OpenEye Python Cookbook** recipe

Constructors

```
OEHasResidueInteractionHint(const OEChem::OEResidue& res)
```

Constructs the functor with the given residue.

```
OEHasResidueInteractionHint(const OEChem::OEResidue& res,
                            const OEInteractionHintComponentTypeBase& ctype)
```

Constructs the functor with the given residue and component type.

```
OEHasResidueInteractionHint(const OEHasResidueInteractionHint& rhs)
```

Copy constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the residue with which the functor is constructed identifies any atom stored in the given interaction. If a component type is defined at construction than that is also checked.

See also

The following built-in component types are available:

- *OELigandInteractionHintComponent*
- *OEProteinInteractionHintComponent*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHasResidueInteractionHint* object is dynamically allocated and owned by the caller.

OEHasSecondaryStructure

```
class OEHasSecondaryStructure : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

A predicate that returns `true` when the atom belongs to a residue that has a secondary structure assignment.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEHasSecondaryStructure(int ssType = OESecondaryStructure::Unassigned)
OEHasSecondaryStructure(const OEHasSecondaryStructure& rhs)
```

Default and copy constructors.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

OEHBondInteractionHint

```
class OEHBondInteractionHint : public OEInteractionHintTypeBase
```

The *OEHBondInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as **hydrogen bond** interactions (inter- and intramolecular).

See also

- *OEHBondInteractionHintType* namespace for specific **hydrogen bond** interaction types
- *OEIsHBondInteractionHint* predicate
- *OEIsIntermolecularHBondInteractionHint* predicate

- *OEIsIntramolecularHBondInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **hydrogen bond** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its methods:
  - *OEPerceiveInteractionOptions.SetMaxAcceptorAngle*
  - *OEPerceiveInteractionOptions.SetMaxChargeAidedHBondDistance*
  - *OEPerceiveInteractionOptions.SetMaxDonorAngle*
  - *OEPerceiveInteractionOptions.SetMaxHBondDistance*
  - *OEPerceiveInteractionOptions.SetMaxUnpairedAngle*
  - *OEPerceiveInteractionOptions.SetMaxUnpairedDistance*
  - *OEPerceiveInteractionOptions.SetMinHBondContactFraction*

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OEHBondInteractionHint(unsigned int type, bool charged = false)
```

type
This value has to be from the *OEHBondInteractionHintType* namespace.

Note

Two *OEHBondInteractionHint* objects with different types from the *OEHBondInteractionHintType* namespace and/or with different charge options are considered to be different.

```
OEHBondInteractionHint(const OEHBondInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEHBondInteractionHint* object is dynamically allocated and owned by the caller.

GetHBondType

```
unsigned int GetHBondType() const
```

Returns the type of the *OEHBondInteractionHint* object. The return value is taken from the *OEHBondInteractionHintType* namespace.

IsCharged

```
bool IsCharged() const
```

Returns whether or not the *OEHBondInteractionHint* object represents a charged interaction.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **hydrogen bond** interaction when added to an *OEInteractionHintContainer* object.

Interactions with the *OEHBondInteractionHint* type can be either **intermolecular** or **intramolecular**.

OEHierChain

```
class OEHierChain
```

This class represents a snapshot of one of the chains inside the molecule used to construct the *OEHierView* object which owns the *OEHierChain* object. It can be used to give a temporary hierarchical view to the underlying molecular data structure.

Constructors

```
OEHierChain()
OEHierChain(OEHierChainImpl* i)
OEHierChain(const OEHierChain &)
```

Default and copy constructors.

GetAtom

```
OEChem::OEAtomBase* GetAtom(const char* resName, int resIdx,
                            unsigned pdbAtomIdx)
const OEChem::OEAtomBase* GetAtom(const char* resName, int resIdx,
                                  unsigned pdbAtomIdx) const
```

Returns an atom specified by the residue name, residue number and atom name.

GetChainID

```
char GetChainID() const
```

Returns the single character representation of the chain ID for this chain of the molecule. This corresponds to the chain ID found in the PDB file.

GetFragments

```
OESystem::OEIterBase<OEHierFragment>* GetFragments()
OESystem::OEIterBase<const OEHierFragment>* GetFragments() const
```

Returns an iterator over all of the fragments in the chain.

GetOEResidue

```
const OEChem::OEResidue& GetOEResidue() const
```

This returns an arbitrary *OEResidue* object associated with this chain.

GetResidue

```
OEHierResidue& GetResidue(const char* resName, int resIdx)
const OEHierResidue& GetResidue(const char* resName, int resIdx) const
```

Gives direct access to a residue using the typical information that a user might know about a residue of interest (*e.g.* - THR242). The chain specifier is not included because the chain is implied by the current object.

OEHierFragment

```
class OEHierFragment
```

The hierarchical view has an *OEHierFragment* object in the hierarchy between *OEHierChain* and *OEHierResidue*. This gives a hierarchical view of a temporary snapshot of the data structure in the molecule used to construct the *OEHierView* which owns the fragment objects.

Constructors

```
OEHierFragment()
OEHierFragment(OEHierFragImpl* i)
OEHierFragment(const OEHierFragment &)
```

Default and copy constructors.

GetAtom

```
OEChem::OEAtomBase* GetAtom(const char* resName, int resIdx,
                            unsigned pdbAtomIdx)
const OEChem::OEAtomBase* GetAtom(const char* resName, int resIdx,
                                  unsigned pdbAtomIdx) const
```

GetFragmentNumber

```
int GetFragmentNumber() const
```

Returns the integer representation of the fragment number for this fragment of the chain. While reading a PDB file, fragment numbers are incremented every time a TER record is encountered or a new ChainID is encountered.

GetOEResidue

```
const OEChem::OEResidue& GetOEResidue() const
```

Returns an arbitrary *OEResidue* object associated with this fragment.

GetResidue

```
OEHierResidue& GetResidue(const char* resName, int resIdx)
const OEHierResidue& GetResidue(const char* resName, int resIdx) const
```

Gives direct access to a residue using the typical information that a user might know about a residue of interest (*e.g.* - THR242). The chain specifier is not included because the chain is implied by the current object.

GetResidues

```
OESystem::OEIterBase<OEHierResidue>* GetResidues()
OESystem::OEIterBase<const OEHierResidue>* GetResidues() const
```

Returns an iterator over all of the residues in the fragment.

OEHierResidue

```
class OEHierResidue
```

This is the lowest temporary object in the hierarchical view. *OEHierResidue* objects are available from within the *OEHierFragment* object and give access to the *OEAtomBase* objects within the molecule at the time the parent *OEHierView* was created. Any atom deletions or additions after the creation of the *OEHierView* will not be available in the object. However, changes to atoms or bonds will be available via the *OEAtomBase* API. The bonds of a molecule are not directly available through the *OEHierView* object, but instead must be accessed via the *OEAtomBase.GetBonds* API point.

Constructors

```
OEHierResidue()
OEHierResidue(OEHierResImpl* i)
OEHierResidue(const OEHierResidue &)
```

Default and copy constructors.

GetAtom

```
OEChem::OEAtomBase* GetAtom(unsigned pdbAtomIdx)
const OEChem::OEAtomBase* GetAtom(unsigned pdbAtomIdx) const
```

Returns an atom specified by the PDB atom index.

GetAtoms

```
OESystem::OEIterBase<OEChem::OEAtomBase>* GetAtoms()
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAtoms() const
```

Returns an iterator over all of the atoms in the residue.

GetOEResidue

```
const OEChem::OEResidue& GetOEResidue() const
```

This returns an arbitrary *OEResidue* object associated with this *OEHierResidue*.

See also

- Example program *ResHist.cs*

GetResidueName

```
const char* GetResidueName() const
```

Returns the residue name.

GetResidueNumber

```
int GetResidueNumber() const
```

Returns the integer representation of the residue number for this residue of the fragment.

OEHierView

```
class OEHierView
```

The *OEHierView* class provides a convenient method for examining biological molecules as if they were hierarchical data. Hierarchical views can be fraught with inefficiencies. Despite this, they are such a common means of thinking about macromolecules that **OEChem** provides access to molecules in this manner. The *OEHierView* class takes a snapshot of an *OEMolBase* and provides convenient access to a hierarchical view of the data.

Note

Unlike typical *OEChem* iterators that closely track changes to a molecule, the hierarchy view is based on a snapshot of the molecule. One should use the assignment operator to update the hierarchical view after any change to the molecule.

Constructors

```
OEHierView()
OEHierView(OEHierViewImpl* i)
OEHierView(const OEHierView &)
```

Default and copy constructors.

```
OEHierView(OEChem::OEMolBase &,
           unsigned assume = OEAssumption::Default)
```

This constructor takes an *OEMolBase* as an argument and takes a snapshot of the molecule and sets up a hierarchical data view of the molecule. This allows hierarchical access to the molecule via the *OEIter* class.

> **Note**
>
> Unlike typical *OEChem* iterators that track changes to a molecule, the hierarchy view is based on a snapshot of the molecule. One should use the assignment operator to update the hierarchical view after any change to the molecule.

GetAtom

```
const OEChem::OEAtomBase * GetAtom(char chainID,
                                   const char* resName, int resIdx,
                                   unsigned pdbAtomIdx)
const OEChem::OEAtomBase* GetAtom(char chainID,
                                  const char* resName, int resIdx,
                                  unsigned pdbAtomIdx) const
```

Returns an atom specified by the chainID, residue name, residue number, and the PDB atom index.

> **See also**
>
> *OEGetPDBAtomIndex* function

GetAtoms

```
OESystem::OEIterBase<OEChem::OEAtomBase>* GetAtoms()
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAtoms() const
```

Returns an iterator over all of the atoms in the hierarchical view.

GetChains

```
OESystem::OEIterBase<OEHierChain>* GetChains()
OESystem::OEIterBase<const OEHierChain>* GetChains() const
```

GetFragments

```
OESystem::OEIterBase<OEHierFragment>* GetFragments()
OESystem::OEIterBase<const OEHierFragment>* GetFragments() const
```

Returns an iterator over all of the fragments in the hierarchical view.

GetResidue

```
OEHierResidue& GetResidue(char chainID, const char* resName,
                          int resIdx)
const OEHierResidue& GetResidue(char chainID, const char* resName,
                                int resIdx) const
```

Gives direct access to a residue using the typical information that a user might know about a residue of interest (*e.g.* - `THR242` of chain `'A'`).

GetResidues

```
OESystem::OEIterBase<OEHierResidue>* GetResidues()
OESystem::OEIterBase<const OEHierResidue>* GetResidues() const
```

Returns an iterator over all of the residues in the hierarchical view.

OEInteractionHint

```
class OEInteractionHint : public OESystem::OEBase
```

The *OEInteractionHint* class is a container that stores a typed interaction between two fragments (*OEInteractionHintFragment*) of an *OEInteractionHintContainer* object.

See also

- *OEInteractionHintContainer* class
- *OEInteractionHintFragment* class
- *OEPerceiveInteractionOptions* class

See also

The following built-in interaction hint types are available:

- *OECationPiInteractionHint*
- *OEChelatorInteractionHint*
- *OEClashInteractionHint*
- *OEContactInteractionHint*
- *OECovalentInteractionHint*
- *OEHBondInteractionHint*
- *OESaltBridgeInteractionHint*
- *OEStackingInteractionHint*
- *OEHalogenBondInteractionHint*

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Depicting Active Site Interactions **Grapheme TK** example

OpenEye Python Cookbook recipes:

- Accessing Interaction Hint Information
- Print Protein-Ligand Interactions to Console
- Print the Summary of Protein-Ligand Interactions for a Set of Molecules

Constructors

```
OEInteractionHint(const OEInteractionHintContainerImpl* container,
                  const OEInteractionHintFragment* fragBgn,
                  const OEInteractionHintFragment* fragEnd,
                  const OEInteractionHintTypeBase& itype)
```

Constructs an *OEInteractionHint* object with the given parameters.

container
The *OEInteractionHintContainer* object the two fragments belong to.

fragBgn, fragEnd
The two fragments of the interaction. The first fragment is going to be considered as the 'begin' fragment, while the second one will be the considered as the 'end' fragment.

type
The type of the interaction that is derived from the *OEInteractionHintTypeBase* abstract base class.

See also

- *OEInteractionHint.GetBgnFragment* method
- *OEInteractionHint.GetEndFragment* method

```
OEInteractionHint(const OEInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OESystem::OEBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory of the returned *OEInteractionHint* object is dynamically allocated and owned by the caller.

HasCalculatedGeometry

```
bool HasCalculatedGeometry(const std::string& name) const
```

Returns `true` if the interaction hint has the geometry calculated with the given name.

name
The name of the calculated geometry. All interaction hint geometry names start with the `bio:interaction:` prefix. The *OEInteractionHint.HasCalculatedGeometry* method can be invoked with or without the `bio:interaction:` prefix in the given name.

See also

- *Accessing Calculated Interaction Hint Geometries* section
- *OEInteractionHint.GetCalculatedGeometry* method
- *OEInteractionHint.GetCalculatedGeometries* method

GetBgnFragment

```
const OEInteractionHintFragment* GetBgnFragment() const
```

Returns the 'begin' fragment of the interaction. The 'begin' fragment is defined when the interaction is constructed.

See also

- *OEInteractionHint.Constructors* method

Note

In case of an "unpaired" interaction both the *OEInteractionHint.GetBgnFragment* and *OEInteractionHint.GetEndFragment* methods will point to the same fragment.

See Also: *OEIsUnpairedInteractionHint* predicate

GetCalculatedGeometry

```
double GetCalculatedGeometry(const std::string& name) const
```

Returns the calculated geometry of the interaction (such as distance or angle). The "distance" values are given in Angstroms, and all "angle" values are given in radians.

name
: The name of the calculated geometry. All interaction hint geometry names start with the `bio:interaction:` prefix. The *OEInteractionHint.GetCalculatedGeometry* method can be invoked with or without the `bio:interaction:` prefix in the given name.

Hint

It is a good practice to call *OEInteractionHint.HasCalculatedGeometry* beforehand to check whether the interaction hint has a geometry with a given name.

Example:

```
if (inter.HasCalculatedGeometry("distance"))
{
    double distance = inter.GetCalculatedGeometry("distance");
    Console.WriteLine("distance: " +
                      String.Format("{0,4:0.000}", distance) + " Angstrom(s)");
}
if (inter.HasCalculatedGeometry("acceptor-angle"))
{
    double angle = ToDegrees(inter.GetCalculatedGeometry("acceptor-angle"));
    Console.WriteLine("acceptor angle: " +
                      String.Format("{0,4:0.000}", angle) + " degree(s)");
}
```

See also

- *Accessing Calculated Interaction Hint Geometries* section
- *OEInteractionHint.HasCalculatedGeometry* method
- *OEInteractionHint.GetCalculatedGeometries* method

GetCalculatedGeometries

```
OESystem::OEIterBase<const std::string>* GetCalculatedGeometries() const
```

Returns an iterator over the geometry names of the interaction hint.

See also

- *Accessing Calculated Interaction Hint Geometries* section
- *OEInteractionHint.HasCalculatedGeometry* method
- *OEInteractionHint.GetCalculatedGeometry* method
- *OEGetCalculatedInteractionGeometries* function

GetDataType

```
const void* GetDataType() const
```

This function is used to perform run-time type identification.

See also

- *OEBase.GetDataType* method in the **OEChem TK** manual

GetDetails

```
std::string GetDetails() const
```

Returns the string data associated with the *OEInteractionHint* object.

GetEndFragment

```
const OEInteractionHintFragment* GetEndFragment() const
```

Returns the ‘end’ fragment of the interaction. The ‘end’ fragment is defined when the interaction is constructed.

See also

- *OEInteractionHint.Constructors* method

Note

In case of an "unpaired" interaction both the *OEInteractionHint.GetBgnFragment* and *OEInteractionHint.GetEndFragment* methods will point to the same fragment.

See Also: *OEIsUnpairedInteractionHint* predicate

GetFragment

```
const OEInteractionHintFragment* GetFragment(const OEInteractionHintComponentTypeBase &)␣
↪const
```

Returns the pointer to one fragment with the given component type. If neither the 'begin' nor the 'end' fragment has the given type, a NULL pointer will be returned.

Warning

In case of an intramolecular interaction, the *OEInteractionHint.GetFragment* can return a NULL pointer, so the return value should always be checked.

See also

The following built-in component types are available:

- *OELigandInteractionHintComponent*
- *OEProteinInteractionHintComponent*

GetInteractionType

```
const OEInteractionHintTypeBase& GetInteractionType() const
```

Returns the interaction type of the *OEInteractionHint* object.

See also

The following built-in fragment connection types are available:

- *OECationPiInteractionHint*
- *OEChelatorInteractionHint*
- *OEClashInteractionHint*
- *OEContactInteractionHint*
- *OECovalentInteractionHint*
- *OEHBondInteractionHint*
- *OESaltBridgeInteractionHint*
- *OEStackingInteractionHint*

- *OEHalogenBondInteractionHint*

GetWeight

```
double GetWeight() const
```

Returns the 'weight' (or score) associated with the *OEInteractionHint* object.

IsDataType

```
bool IsDataType(const void *) const
```

Returns whether type is the same as the instance this method is called on.

See also

- *OEBase.IsDataType* method in the **OEChem TK** manual

IsInter

```
bool IsInter() const
```

Returns whether the two fragments of the *OEInteractionHint* object belongs to the different molecules.

IsIntra

```
bool IsIntra() const
```

Returns whether the two fragments of the *OEInteractionHint* object belongs to the same molecule.

IsValid

```
bool IsValid() const
```

Returns whether the *OEInteractionHint* object represents a valid interaction defined by the type associated with it.

SetDetails

```
void SetDetails(const std::string& details)
```

Sets string data associated with the *OEInteractionHint* object.

SetWeight

```
void SetWeight(double)
```

Sets the 'weight' associated with the *OEInteractionHint* object.

weight
: This number has to be in the range of `[0.0, 1.0]`.

OEInteractionHintComponentTypeBase

```
class OEInteractionHintComponentTypeBase
```

The *OEInteractionHintComponentTypeBase* is an abstract class that enables to type molecule and fragments stored in a *OEInteractionHintContainer* object.

The following classes derive from this class:

- *OELigandInteractionHintComponent*
- *OEProteinInteractionHintComponent*

Constructors

```
OEInteractionHintComponentTypeBase(const std::string& name)
```

Creates an component type with the given name (identifier).

name
A unique string representing a specific component type.

```
OEInteractionHintComponentTypeBase(const OEInteractionHintComponentTypeBase& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintComponentTypeBase* CreateCopy() const = 0
```

Virtual const constructor which allows copying of concrete derived objects using a reference to this base class.

GetName

```
std::string GetName() const
```

Returns the name (identifier) of the connection type.

IsValid

```
bool IsValid(const OEChem::OEMolBase &) const = 0
```

This is a virtual method that has to be implemented in the concrete derived classes to evaluate whether the given molecule is valid for for the given concrete component type.

OEInteractionHintContainer

```
class OEInteractionHintContainer
```

This class represents *OEInteractionHintContainer*, a read only class that stores molecules, the fragments of those molecules (*OEInteractionHintFragment*), and the typed interactions (*OEInteractionHint*) between the fragments.

See also

- *OEInteractionHintFragment* class
- *OEInteractionHint* class
- *OEPerceiveInteractionOptions* class

See also

This class is used by **OEChem TK** and **Grapheme TK** to visualize protein-ligand interactions.

- *OEPerceiveInteractionHints* function in the **OEChem TK** manual
- `OERenderActiveSite` function in the **Grapheme TK** manual
- `OERenderUnpairedInteractionMap` function in the **Grapheme TK** manual

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Depicting Active Site Interactions **Grapheme TK** example
- Accessing Interaction Hint Information **OpenEye Python Cookbook** recipe

Constructors

```
OEInteractionHintContainer()
```

Constructs an empty *OEInteractionHintContainer* object.

```
OEInteractionHintContainer(const OEChem::OEMolBase& protein,
                           const OEChem::OEMolBase& ligand)
```

Initializes an *OEInteractionHintContainer* object with the two given molecules. The first molecule will be added to the container with the *OEProteinInteractionHintComponent* type while the second molecule will be added with the *OELigandInteractionHintComponent* type.

To perceive only intramolecular interactions within one component, pass an empty molecule for the other component. For example, to perceive only intramolecular interactions within the protein, pass an empty molecule as the `ligand` argument. Similarly, to perceive only intramolecular interactions within the ligand, pass an empty molecule as the `protein` argument.

```
OEInteractionHintContainer(const OEBio::OEDesignUnit& du,
                           const unsigned receptorMask =␣
↪OEBio::OEDesignUnitComponents::TargetComplex)
```

Initializes an *OEInteractionHintContainer* from an *OEDesignUnit*. The receptor component is assembled from all design unit components matching `receptorMask` (default: *OEDesignUnitComponents.TargetComplex*). If the design unit has a ligand, it is added with the *OELigandInteractionHintComponent* type. If the design unit has site residues but no ligand, an empty ligand molecule is used so that only intramolecular interactions within the receptor are perceived.

See also

- *OEDesignUnitComponents.TargetComplex* constant
- *OEDesignUnit* class
- *OEConstructInteractionHintContainer* function

operator bool

```
operator bool() const
```

Same as the *OEInteractionHintContainer.IsValid* method.

AddMolecule

```
const OEChem::OEMolBase* AddMolecule(const OEChem::OEMolBase& mol,
                                     const OEInteractionHintComponentTypeBase& itype)
```

Adds a molecule the *OEInteractionHintContainer* with a given type. It returns the pointer of the molecule that is the copied molecule stored inside the interaction container or returns a NULL pointer if adding the molecule with the given type was unsuccessful.

See also

The following built-in component types are available:

- *OELigandInteractionHintComponent*
- *OEProteinInteractionHintComponent*

DeleteInteraction

```
bool DeleteInteraction(const OEInteractionHint* inter)
```

Deletes the given interaction with the corresponding interaction fragments from the *OEInteractionHintContainer* object.

DeleteInteractions

```
bool DeleteInteractions()
```

Deletes all interactions.

GetInteractions

```
OESystem::OEIterBase<const OEInteractionHint>* GetInteractions() const
```

Returns an iterator over all of the interactions stored in the *OEInteractionHintContainer* object.

```
OESystem::OEIterBase<const OEInteractionHint>* GetInteractions(const␣
↪OESystem::OEUnaryPredicate<OEInteractionHint>& pred) const
```

Returns an iterator over all of the interactions of the *OEInteractionHintContainer* object that match the specified interaction predicate.

See also

The following built-in interaction predicates are available:

- *OEHasInteractionHint*
- *OEHasInteractionHintType*
- *OEHasResidueInteractionHint*
- *OEIsCationPiInteractionHint*
- *OEIsChelatorInteractionHint*
- *OEIsClashInteractionHint*
- *OEIsContactInteractionHint*
- *OEIsCovalentInteractionHint*
- *OEIsHalogenBondInteractionHint*
- *OEIsHBondInteractionHint*
- *OEIsInterInteractionHint*
- *OEIsIntermolecularCationPiInteractionHint*
- *OEIsIntermolecularHalogenBondInteractionHint*
- *OEIsIntermolecularHBondInteractionHint*
- *OEIsIntermolecularSaltBridgeInteractionHint*
- *OEIsIntermolecularStackingInteractionHint*
- *OEIsIntraInteractionHint*
- *OEIsIntramolecularCationPiInteractionHint*
- *OEIsIntramolecularChelatorInteractionHint*
- *OEIsIntramolecularHalogenBondInteractionHint*
- *OEIsIntramolecularHBondInteractionHint*
- *OEIsIntramolecularSaltBridgeInteractionHint*
- *OEIsIntramolecularStackingInteractionHint*
- *OEIsNonIdealHBondInteractionHint*
- *OEIsSaltBridgeInteractionHint*
- *OEIsStackingInteractionHint*
- *OEIsClashingHBondInteractionHint*
- *OEIsUnpairedInteractionHint*
- *OEIsUnpairedLigandInteractionHint*
- *OEIsUnpairedProteinInteractionHint*

GetMolecule

```
const OEChem::OEMolBase* GetMolecule(const OEInteractionHintComponentTypeBase& itype)␣
↪const
```

Returns the first molecule stored in the *OEInteractionHintContainer* object with the given type.

See also

The following built-in component types are available:

- *OELigandInteractionHintComponent*
- *OEProteinInteractionHintComponent*

GetMolecules

```
OESystem::OEIterBase<const OEChem::OEMolBase>* GetMolecules() const
```

Returns an iterator over all the molecule of the *OEInteractionHintContainer* object.

```
OESystem::OEIterBase<const OEChem::OEMolBase>* GetMolecules(const␣
↪OEInteractionHintComponentTypeBase& itype) const
```

Returns an iterator over all the molecule with the given type of the *OEInteractionHintContainer* object.

See also

The following built-in component types are available:

- *OELigandInteractionHintComponent*
- *OEProteinInteractionHintComponent*

GetTitle

```
const std::string& GetTitle() const
```

Returns the 'title' of the *OEInteractionHintContainer* object.

HasInteraction

```
bool HasInteraction(const OESystem::OEUnaryPredicate<OEInteractionHint>& pred) const
```

Returns *true* if there is at least one interaction stored in the *OEInteractionHintContainer* object that matches the given predicate.

IsValid

```
bool IsValid() const
```

Returns whether there is at least one molecule that is stored in the *OEInteractionHintContainer* object.

See also

- *OEIsValidActiveSite* function

NumFragments

```
unsigned int NumFragments() const
```

Returns the number of fragments stored in the *OEInteractionHintContainer* object

NumInteractions

```
unsigned int NumInteractions() const
```

Returns the number of interactions stored in the *OEInteractionHintContainer* object.

```
unsigned NumInteractions(const OESystem::OEUnaryPredicate<OEInteractionHint>& pred) const
```

Returns the number of interactions stored in the *OEInteractionHintContainer* object that satisfies the given predicate.

NumMolecules

```
unsigned int NumMolecules() const
```

Returns the number of molecules stored in the *OEInteractionHintContainer* object

SetTitle

```
void SetTitle(const std::string& title)
```

Sets the 'title' of the *OEInteractionHintContainer* object.

OEInteractionHintFragment

```
class OEInteractionHintFragment : public OESystem::OEBase
```

The *OEInteractionHintFragment* class is a container that holds atom pointers of a molecule of a *OEInteractionHintContainer* object.

See also

- *OEInteractionHintContainer* class
- *OEInteractionHint* class

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Depicting Active Site Interactions **Grapheme TK** example

- Accessing Interaction Hint Information **OpenEye Python Cookbook** recipe

Constructors

```
OEInteractionHintFragment(const OEInteractionHintContainerImpl* container,
                          const OEChem::OEMolBase* parent)
```

Creates an *OEInteractionHintFragment* object that belongs to the given container and stores the atoms of the given molecule.

```
OEInteractionHintFragment(const OEInteractionHintFragment& rhs)
```

Copy constructor.

AddAtom

```
bool AddAtom(const OEChem::OEAtomBase *)
```

Adds an atom to the *OEInteractionHintFragment* object. Returns `true` if the atom was successfully added to the *OEInteractionHintFragment* object.

CreateCopy

```
OESystem::OEBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEInteractionHintFragment* object is dynamically allocated and owned by the caller.

GetAtom

```
const OEChem::OEAtomBase* GetAtom(const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>&)␣
↪const
```

Retrieves the first atom of a fragment that matches the specified atom predicate (*OEUnaryPredicate*). If the fragment does not contain an atom that matches, a null pointer is returned.

GetAtoms

```
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAtoms() const
```

Returns an iterator over all the atoms of the fragment.

```
OESystem::OEIterBase<const OEChem::OEAtomBase>* GetAtoms(const OESystem::OEUnaryPredicate
↪<OEChem::OEAtomBase> &) const
```

Returns an iterator over all of the atoms of the fragment that match the specified atom predicate (*OEUnaryPredicate*).

GetComponentType

```
const OEInteractionHintComponentTypeBase& GetComponentType() const
```

Returns the type of the fragment i.e. the type of the molecule the fragment belongs to.

See also

The following built-in component types are available:

- *OELigandInteractionHintComponent*
- *OEProteinInteractionHintComponent*

GetDataType

```
const void* GetDataType() const
```

This function is used to perform run-time type identification.

See also

- *OEBase.GetDataType* method in the **OEChem TK** manual

HasAtom

```
bool HasAtom(const OEChem::OEAtomBase *) const
```

Returns whether the given atom is already stored in the *OEInteractionHintFragment* object.

IsDataType

```
bool IsDataType(const void *) const
```

Returns whether a type is the same as the instance this method is called on.

See also

- *OEBase.IsDataType* method in the **OEChem TK** manual

NumAtoms

```
unsigned int NumAtoms() const
```

Returns the number of atoms of the fragment.

OEInteractionHintTypeBase

```
class OEInteractionHintTypeBase
```

The *OEInteractionHintTypeBase* is an abstract class that enables to type interactions (*OEInteractionHint*) stored in a *OEInteractionHintContainer* object.

The following classes derive from this class:

- *OECationPiInteractionHint*
- *OEChelatorInteractionHint*
- *OEClashInteractionHint*
- *OEContactInteractionHint*
- *OECovalentInteractionHint*
- *OEHalogenBondInteractionHint*
- *OEHBondInteractionHint*
- *OESaltBridgeInteractionHint*
- *OEStackingInteractionHint*

Constructors

```
OEInteractionHintTypeBase(const std::string& name)
```

Creates an interaction type with the given name (identifier).

name
A unique string representing a specific interaction type.

```
OEInteractionHintTypeBase(const OEInteractionHintTypeBase& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const = 0
```

Virtual const constructor which allows copying of concrete derived objects using a reference to this base class.

GetName

```
std::string GetName() const
```

Returns the name (identifier) of the interaction type.

IsValid

```
bool IsValid(const OEInteractionHint &) const
```

Returns whether the given interaction is considered valid for the given interaction type.

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const = 0
```

This is a virtual method that has to be implemented in the concrete derived classes to evaluate whether the two given fragments form a valid interaction for the given concrete interaction type.

OEIridiumData

Attention

This API is currently available in C++ and Python.

```
class OEIridiumData
```

This class contains the Iridium category [Warren-2012], metrics required for categorization, and a methods to determine the Iridium category itself.

See also

- Categorization section in **OESpruce TK**
- *OECalculateDPI* function
- *OEStructureQuality* class
- `OEDrawIridiumData` function in **Grapheme TK** to visualize information stored in an *OEIridiumData* object

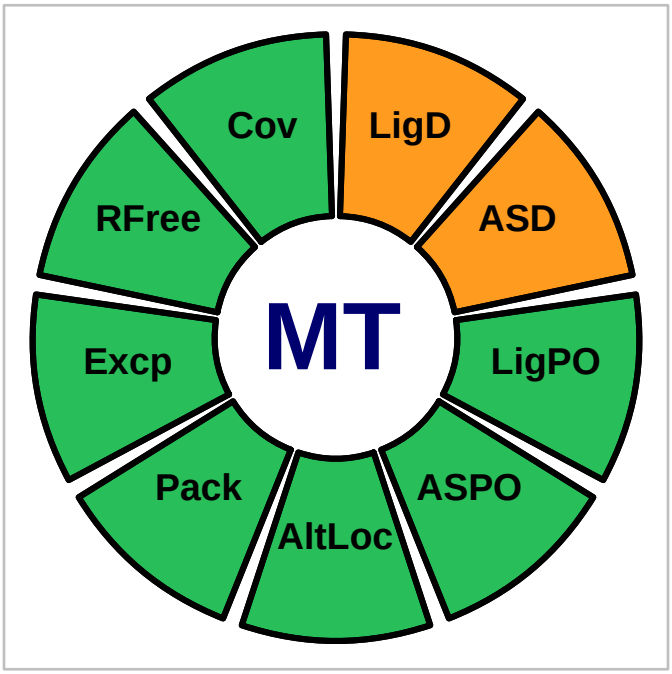


Fig. 30: **Example of visualizing OEIridiumData**

Constructors

```
OEIridiumData()
OEIridiumData(const OEIridiumData& rhs)
```

Default and copy constructor.

operator bool

```
operator bool() const
```

Returns whether the *OEIridiumData* object is valid.

Clear

```
void Clear()
```

Resets the OEIridiumData object to its initial state.

DetermineCategory

```
void DetermineCategory()
```

This function uses the internally stored metrics to determine the Iridium category.

See also

- *OEIridiumCategory* namespace
- Categorization section in **OESpruce TK**

GetAltConfs

```
bool GetAltConfs() const
```

Returns whether or not alternative conformations exist for this structure.

GetAsDensity

```
double GetAsDensity() const
```

Returns the active site density coverage (0 to 1).

GetCategory

```
unsigned GetCategory() const
```

Returns the Iridium category from the *OEIridiumCategory* namespace.

GetDPI

```
double GetDPI() const
```

Returns the diffraction-component precision index (DPI) value.

GetExcipients

```
bool GetExcipients() const
```

Returns whether or not excipients have interactions with the ligand.

GetHasMTZ

```
bool GetHasMTZ() const
```

Returns whether or not an MTZ file exists for this structure.

GetIrrationalRfree

```
bool GetIrrationalRfree() const
```

Returns the irrational Rfree value.

GetLigandDensity

```
double GetLigandDensity() const
```

Returns the ligand density coverage (0 to 1).

GetPackingResidues

```
bool GetPackingResidues() const
```

Returns whether or not packing residues make interactions with the ligand.

GetPartOccupancyLigand

```
bool GetPartOccupancyLigand() const
```

Returns whether or not the ligand has partial occupancy.

GetPartOccupancyProtein

```
bool GetPartOccupancyProtein() const
```

Returns whether or not the protein has partial occupancy.

GetPossibleCovalent

```
bool GetPossibleCovalent() const
```

Returns whether or not potential but unperceived covalent interaction can be found in the active site.

GetRFree

```
double GetRFree() const
```

Returns the Rfree value.

GetResolution

```
double GetResolution() const
```

Returns the resolution of the crystal structure.

SetAltConfs

```
bool SetAltConfs(bool altConfs)
```

Sets whether or not alternative conformations exist for this structure.

SetAsDensity

```
bool SetAsDensity(double asDensity)
```

Sets the active site density coverage.

SetCategory

```
bool SetCategory(unsigned category)
```

Sets the Iridium category from a value in the *OEIridiumCategory* namespace.

SetDPI

```
bool SetDPI(double dpi)
```

Sets the diffraction-component precision index (DPI) value.

SetExcipients

```
bool SetExcipients(bool excipients)
```

Sets whether or not excipients have interactions with the ligand.

SetHasMTZ

```
bool SetHasMTZ(bool hasMTZ)
```

Sets whether or not an MTZ file exists for this structure.

SetIrrationalRfree

```
bool SetIrrationalRfree(bool irrationalRfree)
```

Sets the irrational Rfree value.

SetLigandDensity

```
bool SetLigandDensity(double ligandDensity)
```

Sets the ligand density coverage (0 to 1).

SetPackingResidues

```
bool SetPackingResidues(bool packingResidues)
```

Sets whether or not packing residues make interactions with the ligand.

SetPartOccupancyLigand

```
bool SetPartOccupancyLigand(bool partOccupancyLigand)
```

Sets whether or not the ligand has partial occupancy.

SetPartOccupancyProtein

```
bool SetPartOccupancyProtein(bool partOccupancyProtein)
```

Sets whether or not the protein has partial occupancy.

SetPossibleCovalent

```
bool SetPossibleCovalent(bool possibleCovalent)
```

Sets whether or not potential but unperceived covalent interaction can be found in the active site.

SetRFree

```
bool SetRFree(double rFree)
```

Sets the Rfree value.

SetResolution

```
bool SetResolution(double resolution)
```

Sets the resolution of the crystal structure.

OEIsAlphaCarbon

```
class OEIsAlphaCarbon : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

Warning

This is a deprecated functor. Use *OEIsCAlpha* predicate instead.

OEIsBackboneAtom

```
class OEIsBackboneAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsBackboneAtom* functor that identifies atoms (*OEAtomBase*) that are part of a macro-molecule backbone.

See also

- *Residue Data Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsBackboneAtom(bool considercarbonyl = true,
                 bool considerterminaloxygen = false)
```

Default constructor.

considercarbonyl
Determines whether to consider the carbonyl oxygen as a backbone atom.

considerterminaloxygen
Determines whether to consider the terminal oxygen (OXT) as a backbone atom.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsBackboneAtom* object is dynamically allocated and owned by the caller.

OEIsCationPiInteractionHint

```
class OEIsCationPiInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsCationPiInteractionHint* functor that identifies **cation-Pi** interactions.

See also

- *OECationPiInteractionHint* class
- *OECationPiInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsIntermolecularCationPiInteractionHint* predicate
- *OEIsIntramolecularCationPiInteractionHint* predicate

Constructors

```
OEIsCationPiInteractionHint(bool includeIntra = true)
```

Constructor.

param includeIntra
If `true` (default), intramolecular cation-pi interactions are included in the results.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has a cation-pi type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OECationPiInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsCationPiInteractionHint* object is dynamically allocated and owned by the caller.

OEIsChelatorInteractionHint

```
class OEIsChelatorInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsChelatorInteractionHint* functor that identifies **chelator** interactions.

See also

- *OEChelatorInteractionHint* class
- *OEChelatorInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsIntramolecularChelatorInteractionHint* predicate

Constructors

```
OEIsChelatorInteractionHint(bool includeIntra = true)
```

Constructor.

param includeIntra
If `true` (default), intramolecular chelator interactions are included in the results.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has a chelator type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEChelatorInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsChelatorInteractionHint* object is dynamically allocated and owned by the caller.

OEIsClashingHBondInteractionHint

```
class OEIsClashingHBondInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsClashingHBondInteractionHint* functor that identifies **clashing** hydrogen bond interactions, i.e. geometrically unfavorable interactions between two acceptors or two donors that cannot form a proper hydrogen bond.

See also

- *OEHBondInteractionHint* class
- *OEHBondInteractionHintType* namespace

- *OEInteractionHint* class
- *OEIsHBondInteractionHint* predicate
- *OEIsNonIdealHBondInteractionHint* predicate
- *OEIsUnpairedInteractionHint* predicate

Constructors

```
OEIsClashingHBondInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has a clashing hydrogen bond type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHBondInteractionHint* with one of the following types:

- *OEHBondInteractionHintType.ClashAcceptorAcceptor*
- *OEHBondInteractionHintType.ClashDonorDonor*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsClashingHBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsClashInteractionHint

```
class OEIsClashInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsClashInteractionHint* functor that identifies **clash** interactions.

See also

- *OEClashInteractionHint* class
- *OEInteractionHint* class

Constructors

```
OEIsClashInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has a clash type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEClashInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsClashInteractionHint* object is dynamically allocated and owned by the caller.

OEIsContactInteractionHint

```
class OEIsContactInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsContactInteractionHint* functor that identifies **contact** interactions.

See also

- *OEContactInteractionHint* class
- *OEInteractionHint* class

Constructors

```
OEIsContactInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has a contact type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEContactInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsContactInteractionHint* object is dynamically allocated and owned by the caller.

OEIsCovalentInteractionHint

```
class OEIsCovalentInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsCovalentInteractionHint* functor that identifies **covalent** interactions.

See also

- *OECovalentInteractionHint* class
- *OEInteractionHint* class

Constructors

```
OEIsCovalentInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has a clash type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OECovalentInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsCovalentInteractionHint* object is dynamically allocated and owned by the caller.

OEIsCTerminalAtom

```
class OEIsCTerminalAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsCTerminalAtom* functor that identifies the C-terminal backbone atoms (*OEAtomBase*) for standard amino acid residues in a macro-molecule, or for a nonstandard amino acid residues with a complete backbone.

See also

- *Residue Data Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsCTerminalAtom()
```

Default constructor.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsCTerminalAtom* object is dynamically allocated and owned by the caller.

OEIsDisulfideBond

```
class OEIsDisulfideBond : public OESystem::OEUnaryPredicate<OEChem::OEBondBase>
```

This class represents *OEIsDisulfideBond* functor that identifies the disulfide bonds.

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsDisulfideBond()
```

Default constructor.

operator()

```
bool operator()(const OEChem::OEBondBase& bond) const
```

Returns `true` if the bond is disulfide; otherwise, it returns `false`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEBondBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsDisulfideBond* object is dynamically allocated and owned by the caller.

OEIsHalogenBondInteractionHint

```
class OEIsHalogenBondInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsHalogenBondInteractionHint* functor that identifies **halogen bonding** interactions.

See also

- *OEHalogenBondInteractionHint* class
- *OEHalogenBondInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsIntermolecularHalogenBondInteractionHint* predicate
- *OEIsIntramolecularHalogenBondInteractionHint* predicate

Constructors

```
OEIsHalogenBondInteractionHint(bool includeIntra = true)
```

Constructor.

param includeIntra
If `true` (default), intramolecular halogen bond interactions are included in the results.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has a halogen bonding type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHalogenBondInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHalogenBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsHBondInteractionHint

```
class OEIsHBondInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsHBondInteractionHint* functor that identifies **hydrogen bonding** interactions.

See also

- *OEHBondInteractionHint* class
- *OEHBondInteractionHintType* namespace

- *OEInteractionHint* class
- *OEIsIntermolecularHBondInteractionHint* predicate
- *OEIsIntramolecularHBondInteractionHint* predicate
- *OEIsNonIdealHBondInteractionHint* predicate
- *OEIsClashingHBondInteractionHint* predicate
- *OEIsUnpairedInteractionHint* predicate

Constructors

```
OEIsHBondInteractionHint(bool includeIntra = true,
                         bool includeNonIdeal = true)
```

Constructor.

param includeIntra
If `true` (default), intramolecular hydrogen bond interactions are included in the results.

param includeNonIdeal
If `true` (default), non-ideal hydrogen bond interactions (long donor-acceptor distances or poor geometries) are included in the results.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has a hydrogen bonding type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHBondInteractionHint* type. Clashing hydrogen bond interactions (*OEHBondInteractionHintType.ClashAcceptorAcceptor*, *OEHBondInteractionHintType.ClashDonorDonor*) and unpaired interactions (*OEHBondInteractionHintType.UnpairedLigandDonor*, *OEHBondInteractionHintType.UnpairedLigandAcceptor*, *OEHBondInteractionHintType.UnpairedProteinDonor*, *OEHBondInteractionHintType.UnpairedProteinAcceptor*) are **not** included. Use *OEIsClashingHBondInteractionHint* and *OEIsUnpairedInteractionHint* to retrieve those interaction types.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsHetAtom

```
class OEIsHetAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsHetAtom* functor that identifies the atoms (*OEAtomBase*) that have the 'hetero atom' property in a residue.

See also

- *OEResidue.IsHetAtom* method

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsHetAtom()
```

Default constructor.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

Returns `true` if the `atom` has the 'hetero atom' property in a residue; otherwise, it returns `false`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsHetAtom* object is dynamically allocated and owned by the caller.

OEIsInterInteractionHint

```
class OEIsInterInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsInterInteractionHint* functor that identifies connections (*OEInteractionHint*) within two different molecules stored in the *OEInteractionHintContainer* object.

See also

- *OEInteractionHintContainer.GetInteractions* method
- *OEIsIntraInteractionHint* functor
- *OEInteractionHint.IsInter* method

Constructors

```
OEIsInterInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the interaction is between two different molecules.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsInterInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntermolecularCationPiInteractionHint

```
class OEIsIntermolecularCationPiInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntermolecularCationPiInteractionHint* functor that identifies **cation-pi** interactions between atoms in different molecules (i.e. between the ligand and the protein).

See also

- *OECationPiInteractionHint* class
- *OECationPiInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsCationPiInteractionHint* predicate
- *OEIsIntramolecularCationPiInteractionHint* predicate

Constructors

```
OEIsIntermolecularCationPiInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intermolecular cation-pi type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OECationPiInteractionHint* with one of the following types:

- *OECationPiInteractionHintType.LigandPi*
- *OECationPiInteractionHintType.ProteinPi*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntermolecularCationPiInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntermolecularChelatorInteractionHint

```
class OEIsIntermolecularChelatorInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntermolecularChelatorInteractionHint* functor that identifies **chelator** interactions between two atoms in the different molecule.

See also

- *OEChelatorInteractionHint* class
- *OEChelatorInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsChelatorInteractionHint* predicate
- *OEIsIntramolecularChelatorInteractionHint* predicate

Constructors

```
OEIsIntermolecularChelatorInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has a hydrogen bonding type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEChelatorInteractionHint* with the following types:

- *OEChelatorInteractionHintType.LigandChelates*
- *OEChelatorInteractionHintType.ProteinChelates*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntermolecularChelatorInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntermolecularHalogenBondInteractionHint

```
class OEIsIntermolecularHalogenBondInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntermolecularHalogenBondInteractionHint* functor that identifies **halogen bond** interactions between atoms in different molecules (i.e. between the ligand and the protein).

See also

- *OEHalogenBondInteractionHint* class
- *OEHalogenBondInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsHalogenBondInteractionHint* predicate
- *OEIsIntramolecularHalogenBondInteractionHint* predicate

Constructors

```
OEIsIntermolecularHalogenBondInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intermolecular halogen bond type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHalogenBondInteractionHint* with one of the following types:

- *OEHalogenBondInteractionHintType.LigandElectrophile*
- *OEHalogenBondInteractionHintType.LigandNucleophile*
- *OEHalogenBondInteractionHintType.ProteinElectrophile*
- *OEHalogenBondInteractionHintType.ProteinNucleophile*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntermolecularHalogenBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntermolecularHBondInteractionHint

```
class OEIsIntermolecularHBondInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntermolecularHBondInteractionHint* functor that identifies **hydrogen bonding** interactions between two atom in different molecules.

See also

- *OEHBondInteractionHint* class
- *OEHBondInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsHBondInteractionHint* predicate
- *OEIsIntramolecularHBondInteractionHint* predicate
- *OEIsClashingHBondInteractionHint* predicate

Constructors

```
OEIsIntermolecularHBondInteractionHint(bool includeNonIdeal = true)
```

Constructor.

param includeNonIdeal
If `true` (default), non-ideal hydrogen bond interactions (long donor-acceptor distances or poor geometries) are included in the results.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has a hydrogen bonding type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHBondInteractionHint* with the following types:

- *OEHBondInteractionHintType.LigandAccepts*
- *OEHBondInteractionHintType.LigandDonates*

When `includeNonIdeal` is `true` (default), the following types are also included:

- *OEHBondInteractionHintType.NonIdealLigandAccepts*
- *OEHBondInteractionHintType.NonIdealLigandDonates*

Clashing interactions (*OEHBondInteractionHintType.ClashAcceptorAcceptor*, *OEHBondInteractionHintType.ClashDonorDonor*) are not included. Use *OEIsClashingHBondInteractionHint* for those.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntermolecularHBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntermolecularSaltBridgeInteractionHint

```
class OEIsIntermolecularSaltBridgeInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntermolecularSaltBridgeInteractionHint* functor that identifies **salt-bridge** interactions between atoms in different molecules (i.e. between the ligand and the protein).

See also

- *OESaltBridgeInteractionHint* class
- *OESaltBridgeInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsSaltBridgeInteractionHint* predicate
- *OEIsIntramolecularSaltBridgeInteractionHint* predicate

Constructors

```
OEIsIntermolecularSaltBridgeInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intermolecular salt-bridge type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OESaltBridgeInteractionHint* with one of the following types:

- *OESaltBridgeInteractionHintType.LigandNegative*
- *OESaltBridgeInteractionHintType.LigandPositive*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntermolecularSaltBridgeInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntermolecularStackingInteractionHint

```
class OEIsIntermolecularStackingInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntermolecularStackingInteractionHint* functor that identifies **stacking** interactions between aromatic rings in different molecules (i.e. between the ligand and the protein).

See also

- *OEStackingInteractionHint* class
- *OEStackingInteractionHintType* namespace
- *OEInteractionHint* class

- *OEIsStackingInteractionHint* predicate
- *OEIsIntramolecularStackingInteractionHint* predicate

Constructors

```
OEIsIntermolecularStackingInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intermolecular stacking type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEStackingInteractionHint* with one of the following types:

- *OEStackingInteractionHintType.Pi*
- *OEStackingInteractionHintType.T*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntermolecularStackingInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntraInteractionHint

```
class OEIsIntraInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsIntraInteractionHint* functor that identifies connections (*OEInteractionHint*) within the same molecule stored in the *OEInteractionHintContainer* object.

See also

- *OEInteractionHintContainer.GetInteractions* method
- *OEIsInterInteractionHint* functor
- *OEInteractionHint.IsIntra* method

Constructors

```
OEIsIntraInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the interaction is within the same molecule.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntraInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntramolecularCationPiInteractionHint

```
class OEIsIntramolecularCationPiInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntramolecularCationPiInteractionHint* functor that identifies **intramolecular cation-pi** interactions between a cation and an aromatic ring within the same molecule (ligand or protein).

See also

- *OECationPiInteractionHint* class
- *OECationPiInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsCationPiInteractionHint* predicate
- *OEIsIntermolecularCationPiInteractionHint* predicate

Constructors

```
OEIsIntramolecularCationPiInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intramolecular cation-pi type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OECationPiInteractionHint* with one of the following types:

- *OECationPiInteractionHintType.IntramolecularLigand*
- *OECationPiInteractionHintType.IntramolecularProtein*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntramolecularCationPiInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntramolecularChelatorInteractionHint

```
class OEIsIntramolecularChelatorInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntramolecularChelatorInteractionHint* functor that identifies **chelator** interactions between two atoms in the same molecule.

See also

- *OEChelatorInteractionHint* class
- *OEChelatorInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsChelatorInteractionHint* predicate
- *OEIsIntermolecularChelatorInteractionHint* predicate

Constructors

```
OEIsIntramolecularChelatorInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has a chelator bonding type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEChelatorInteractionHint* with *OEChelatorInteractionHintType.IntramolecularProtein* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntramolecularChelatorInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntramolecularHalogenBondInteractionHint

```
class OEIsIntramolecularHalogenBondInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntramolecularHalogenBondInteractionHint* functor that identifies **intramolecular halogen bond** interactions between a halogen and an acceptor within the same molecule (ligand or protein).

See also

- *OEHalogenBondInteractionHint* class
- *OEHalogenBondInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsHalogenBondInteractionHint* predicate
- *OEIsIntermolecularHalogenBondInteractionHint* predicate

Constructors

```
OEIsIntramolecularHalogenBondInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intramolecular halogen bond type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHalogenBondInteractionHint* with one of the following types:

- *OEHalogenBondInteractionHintType.IntramolecularLigand*
- *OEHalogenBondInteractionHintType.IntramolecularProtein*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntramolecularHalogenBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntramolecularHBondInteractionHint

```
class OEIsIntramolecularHBondInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntramolecularHBondInteractionHint* functor that identifies **hydrogen bonding** interactions between two atoms in the same molecule.

See also

- *OEHBondInteractionHint* class
- *OEHBondInteractionHintType* namespace
- *OEInteractionHint* class

- *OEIsHBondInteractionHint* predicate
- *OEIsIntermolecularHBondInteractionHint* predicate
- *OEIsUnpairedInteractionHint* predicate

Constructors

```
OEIsIntramolecularHBondInteractionHint(bool includeNonIdeal = true)
```

Constructor.

param includeNonIdeal
If `true` (default), non-ideal intramolecular hydrogen bond interactions are included in the results.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has an intramolecular hydrogen bonding type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHBondInteractionHint* with the following types:

- *OEHBondInteractionHintType.IntramolecularLigand*
- *OEHBondInteractionHintType.IntramolecularProtein*

When `includeNonIdeal` is `true` (default), the following types are also included:

- *OEHBondInteractionHintType.NonIdealProteinAccepts*
- *OEHBondInteractionHintType.NonIdealProteinDonates*
- *OEHBondInteractionHintType.NonIdealLigandDonates*
- *OEHBondInteractionHintType.NonIdealLigandAccepts*

Unpaired interactions are not included. Use *OEIsUnpairedInteractionHint* to retrieve unpaired hydrogen bond donors and acceptors.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntramolecularHBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntramolecularSaltBridgeInteractionHint

```
class OEIsIntramolecularSaltBridgeInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntramolecularSaltBridgeInteractionHint* functor that identifies **intramolecular salt-bridge** interactions between oppositely charged groups within the same molecule (ligand or protein).

See also

- *OESaltBridgeInteractionHint* class
- *OESaltBridgeInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsSaltBridgeInteractionHint* predicate
- *OEIsIntermolecularSaltBridgeInteractionHint* predicate

Constructors

```
OEIsIntramolecularSaltBridgeInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intramolecular salt-bridge type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OESaltBridgeInteractionHint* with one of the following types:

- *OESaltBridgeInteractionHintType.IntramolecularLigand*
- *OESaltBridgeInteractionHintType.IntramolecularProtein*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntramolecularSaltBridgeInteractionHint* object is dynamically allocated and owned by the caller.

OEIsIntramolecularStackingInteractionHint

```
class OEIsIntramolecularStackingInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsIntramolecularStackingInteractionHint* functor that identifies **intramolecular stacking** interactions between aromatic rings within the same molecule (ligand or protein).

See also

- *OEStackingInteractionHint* class
- *OEStackingInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsStackingInteractionHint* predicate
- *OEIsIntermolecularStackingInteractionHint* predicate

Constructors

```
OEIsIntramolecularStackingInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has an intramolecular stacking type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEStackingInteractionHint* with one of the following types:

- *OEStackingInteractionHintType.IntramolecularLigandPi*
- *OEStackingInteractionHintType.IntramolecularLigandT*
- *OEStackingInteractionHintType.IntramolecularProteinPi*
- *OEStackingInteractionHintType.IntramolecularProteinT*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsIntramolecularStackingInteractionHint* object is dynamically allocated and owned by the caller.

OEIsNonIdealHBondInteractionHint

```
class OEIsNonIdealHBondInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsNonIdealHBondInteractionHint* functor that identifies **non-ideal hydrogen bond** (i.e. hydrogen bonds with long donor-acceptor distances or poor geometries) interactions.

See also

- *OEHBondInteractionHint* class
- *OEHBondInteractionHintType* namespace
- *OEInteractionHint* class
- *OEPerceiveInteractionOptions* class
- *OEIsHBondInteractionHint* predicate

Constructors

```
OEIsNonIdealHBondInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& c) const
```

Returns `true`, if the given interaction has non-ideal hydrogen bonding type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEHBondInteractionHint* with the following types:

- *OEHBondInteractionHintType.NonIdealLigandAccepts*
- *OEHBondInteractionHintType.NonIdealLigandDonates*
- *OEHBondInteractionHintType.NonIdealProteinAccepts*
- *OEHBondInteractionHintType.NonIdealProteinDonates*

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsNonIdealHBondInteractionHint* object is dynamically allocated and owned by the caller.

OEIsNTerminalAtom

```
class OEIsNTerminalAtom : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsNTerminalAtom* functor that identifies the N-terminal backbone atoms (*OEAtomBase*) for standard amino acid residues in a macro-molecule, or for a nonstandard amino acid residues with a complete backbone.

See also

- *Residue Data Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsNTerminalAtom()
```

Default constructor.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsNTerminalAtom* object is dynamically allocated and owned by the caller.

OEIsNucleicAcidBase

```
class OEIsNucleicAcidBase : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsNucleicAcidBase* functor that identifies nucleic acid base atoms (*OEAtomBase*).

See also

- *OEGetPDBAtomIndex* function
- *Residue Data Functors* section
- *OEIsNucleicAcidSugar* predicate
- *OEIsNucleicAcidPhosphate* predicate

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

Returns `true`, if the atom read from a pdb file with the following pdb atom names:

- *OEPDBAtomName.C2*, *OEPDBAtomName.C5*, *OEPDBAtomName.C6*, *OEPDBAtomName.C8*
- *OEPDBAtomName.N1*, *OEPDBAtomName.N2*, *OEPDBAtomName.N3*, *OEPDBAtomName.N4*, *OEPDBAtomName.N6*, *OEPDBAtomName.N7*, *OEPDBAtomName.N9*
- *OEPDBAtomName.O2*, *OEPDBAtomName.O4*, *OEPDBAtomName.O6*

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsNucleicAcidBase* object is dynamically allocated and owned by the caller.

OEIsNucleicAcidPhosphate

```
class OEIsNucleicAcidPhosphate : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsNucleicAcidPhosphate* functor that identifies nucleic acid phosphate atoms (*OEAtomBase*).

See also

- *OEGetPDBAtomIndex* function
- *Residue Data Functors* section
- *OEIsNucleicAcidBase* predicate
- *OEIsNucleicAcidSugar* predicate

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

Returns `true`, if the atom read from a `pdb` file with the following pdb atom names:

- *OEPDBAtomName.P*
- *OEPDBAtomName.O1P*, *OEPDBAtomName.O2P*, *OEPDBAtomName.O3P*

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsNucleicAcidPhosphate* object is dynamically allocated and owned by the caller.

OEIsNucleicAcidSugar

```
class OEIsNucleicAcidSugar : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsNucleicAcidSugar* functor that identifies nucleic acid sugar atoms (*OEAtomBase*).

See also

- *OEGetPDBAtomIndex* function
- *Residue Data Functors* section

- *OEIsNucleicAcidBase* predicate
- *OEIsNucleicAcidPhosphate* predicate

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

Returns `true`, if the atom read from a pdb file with the following pdb atom names:

- *OEPDBAtomName.O2_*, *OEPDBAtomName.O3_*, *OEPDBAtomName.O4_*, *OEPDBAtomName.O5_*,
- *OEPDBAtomName.C1_*, *OEPDBAtomName.C2_*, *OEPDBAtomName.C3_*, *OEPDBAtomName.C4_*, *OEPDBAtomName.C5_*

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsNucleicAcidSugar* object is dynamically allocated and owned by the caller.

OEIsSaltBridgeInteractionHint

```
class OEIsSaltBridgeInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsSaltBridgeInteractionHint* functor that identifies **salt-bridge** interactions.

See also

- *OESaltBridgeInteractionHint* class
- *OESaltBridgeInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsIntermolecularSaltBridgeInteractionHint* predicate
- *OEIsIntramolecularSaltBridgeInteractionHint* predicate
- *OEIsUnpairedInteractionHint* predicate

Constructors

```
OEIsSaltBridgeInteractionHint(bool includeIntra = true)
```

Constructor.

param includeIntra
If `true` (default), intramolecular salt-bridge interactions are included in the results.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has a salt-bridge type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OESaltBridgeInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsSaltBridgeInteractionHint* object is dynamically allocated and owned by the caller.

OEIsStackingInteractionHint

```
class OEIsStackingInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsStackingInteractionHint* functor that identifies **stacking** interactions.

See also

- *OEStackingInteractionHint* class
- *OEStackingInteractionHintType* namespace
- *OEInteractionHint* class
- *OEIsIntermolecularStackingInteractionHint* predicate
- *OEIsIntramolecularStackingInteractionHint* predicate

Constructors

```
OEIsStackingInteractionHint(bool includeIntra = true)
```

Constructor.

param includeIntra
If `true` (default), intramolecular stacking interactions are included in the results.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has a stacking type *i.e.* the *OEInteractionHint.GetInteractionType* method returns an *OEStackingInteractionHint* type.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsStackingInteractionHint* object is dynamically allocated and owned by the caller.

OEIsStandardAminoAcid

```
class OEIsStandardAminoAcid : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsStandardAminoAcid* functor that identifies atoms (*OEAtomBase*) that are in standard amino acid residues

See also

- *Residue Data Functors* section
- `OEIsStandardProteinResidue()` function

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsStandardAminoAcid()
```

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsStandardAminoAcid* object is dynamically allocated and owned by the caller.

OEIsStandardNucleicAcid

```
class OEIsStandardNucleicAcid : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsStandardNucleicAcid* functor that identifies atoms (*OEAtomBase*) that are in standard nucleic acid residues

See also

- *Residue Data Functors* section
- `OEIsStandardNucleicAcidResidue()` function

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsStandardNucleicAcid()
```

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase , bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsStandardNucleicAcid* object is dynamically allocated and owned by the caller.

OEIsUnpairedInteractionHint

```
class OEIsUnpairedInteractionHint : public OESystem::OEUnaryPredicate<OEInteractionHint>
```

This class represents *OEIsUnpairedInteractionHint* functor that identifies **unpaired** interactions — hydrogen bond donors or acceptors, and charged groups in salt-bridge positions, that have no complementary partner in the active site.

Note

This predicate covers both unpaired hydrogen bond interactions (*OEHBondInteractionHintType.UnpairedLigandDonor*, *OEHBondInteractionHintType.UnpairedLigandAcceptor*,

OEHBondInteractionHintType.UnpairedProteinDonor, *OEHBondInteractionHintType.UnpairedProteinAcceptor*) and unpaired salt-bridge interactions (*OESaltBridgeInteractionHintType.UnpairedLigandPositive*, *OESaltBridgeInteractionHintType.UnpairedLigandNegative*, *OESaltBridgeInteractionHintType.UnpairedProteinPositive*, *OESaltBridgeInteractionHintType.UnpairedProteinNegative*).

See also

- *OEInteractionHint* class
- *OEIsUnpairedLigandInteractionHint* predicate
- *OEIsUnpairedProteinInteractionHint* predicate
- *OEIsHBondInteractionHint* predicate
- *OEIsSaltBridgeInteractionHint* predicate

Constructors

```
OEIsUnpairedInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction is an unpaired hydrogen bond or unpaired salt-bridge interaction. This is equivalent to *OEIsUnpairedLigandInteractionHint* || *OEIsUnpairedProteinInteractionHint*.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsUnpairedInteractionHint* object is dynamically allocated and owned by the caller.

OEIsUnpairedLigandInteractionHint

```
class OEIsUnpairedLigandInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsUnpairedLigandInteractionHint* functor that identifies interactions with only ligand component.

See also

- *OEInteractionHint* class

Constructors

```
OEIsUnpairedLigandInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has only a ligand component.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsUnpairedLigandInteractionHint* object is dynamically allocated and owned by the caller.

OEIsUnpairedProteinInteractionHint

```
class OEIsUnpairedProteinInteractionHint : public OESystem::OEUnaryPredicate
↪<OEInteractionHint>
```

This class represents *OEIsUnpairedProteinInteractionHint* functor that identifies interactions with only protein component..

See also

- *OEInteractionHint* class

Constructors

```
OEIsUnpairedProteinInteractionHint()
```

Default constructor.

operator()

```
bool operator()(const OEInteractionHint& i) const
```

Returns `true`, if the given interaction has only a protein component.

CreateCopy

```
OESystem::OEUnaryFunction<OEInteractionHint, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsUnpairedProteinInteractionHint* object is dynamically allocated and owned by the caller.

OEIsWater

```
class OEIsWater : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsWater* functor that determines if an atom (*OEAtomBase*) is in a water molecule.

See also

- *Residue Data Functors* section

The following methods are publicly inherited from *OEUnaryPredicate*:

operator()	*CreateCopy*	*CreatePredicateCopy*

The following methods are publicly inherited from *OEUnaryFunction*:

operator()	*CreateCopy*

Constructors

```
OEIsWater(bool checkHydrogens = false, bool allowIons = false)
```

Constructs the functor to determine if an atom is part of a water molecule. If the optional argument `checkHydrogens` is set to `true`, the function checks whether hydrogen atoms are part of a water molecule and returns `true`. The default behavior only returns `true` for water oxygen atoms, which is useful for counting water molecules in a system. If optional argument `allowIons` is set to `true`, the function allows checking for the water ions OH^- and H_3O^+.

operator()

```
bool operator()(const OEChem::OEAtomBase& atom) const
```

Returns `true`, if the atom

- read from a pdb file with the following pdb residue names: *HOH*, *H2O*, *DOD*, *D2O*, *WAT*, *TIP*, *SOL*, *OH2*, *OD2*.
- is an oxygen and has no heavy atom neighbor

otherwise it returns `false`.

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool>* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEIsWater* object is dynamically allocated and owned by the caller.

OELigandInteractionHintComponent

```
class OELigandInteractionHintComponent : public OEInteractionHintComponentTypeBase
```

Built-in component type that classifies molecules as being 'ligand' when stored in a *OEInteractionHintContainer* object.

The following methods are publicly inherited from *OEInteractionHintComponentTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OELigandInteractionHintComponent()
```

Default constructor.

```
OELigandInteractionHintComponent(const OELigandInteractionHintComponent& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintComponentTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OELigandInteractionHintComponent* object is dynamically allocated and owned by the caller.

GetMaxHeavyAtom

```
unsigned int GetMaxHeavyAtom() const
```

Returns the maximum number of heavy atoms that a molecule allowed to have in order to be able to be considered as a 'ligand' in a *OEInteractionHintContainer* object.

IsValid

```
bool IsValid(const OEChem::OEMolBase &) const
```

Returns whether a molecule can be inserted to a *OEInteractionHintContainer* object with a *OELigandInteractionHintComponent* type *i.e.* it returns true if the molecule has 3D coordinates and its number of heavy atoms in the range of [1, 200].

See also

- *OELigandInteractionHintComponent.SetMaxHeavyAtom* method
- *OELigandInteractionHintComponent.GetMaxHeavyAtom* method

SetMaxHeavyAtom

```
void SetMaxHeavyAtom(unsigned int maxhvy)
```

Sets the maximum number of heavy atoms that a molecule allowed to have in order to be able to be considered as a 'ligand' in a *OEInteractionHintContainer* object.

OEMolComplexCategorizer

```
class OEMolComplexCategorizer : public OEMolComplexCategorizerBase
```

Concrete class functor used by *OESplitMolComplex* and *OEGetMolComplexComponents* to assign functional roles to molecular fragments. The *OESplitMolComplexOptions* class holds a *OEMolComplexCategorizer* object by default.

The following methods are publicly inherited from *OEMolComplexCategorizerBase*:

operator()	*GetBundleCofactors*
CreateCopy	*UsingExplicitLigandName*
GetResidueCategoryData	

Constructors

```
OEMolComplexCategorizer(const OEMolComplexCategorizer& rhs)
OEMolComplexCategorizer(const std::string& ligName = std::string())
```

Constructor and copy constructor. If `ligName` is specified in the call to the constructor, the method *UsingExplicitLigandName* will return `true` and components will be recognized as ligands if and only if they have a matching residue name string.

See also

- *OEMolComplexCategorizer.AddLigandName*

operator()

```
std::string operator() (OESystem::OERoleSet& roles, OEMolComplexResCounts& counts, const␣
↪OEChem::OEAtomBondSet& frag, bool isCovalent = false) const
```

Method to examine the *OEAtomBondSet* `frag` and return three things required by *OESplitMolComplex* and *OEGetMolComplexComponents* to process the component:

- a new title string
- a *OERoleSet* containing the functional roles associated with this fragment (see *OEMolComplexFilterCategory*)
- an updated map of how many times a residue has been seen to date

If `isCovalent` is true, additional tests may be performed to determine whether the fragment might be considered a (covalent) ligand which is indicated with the *OEMolComplexFilterCategory.Ligand* *OERole* in `roles`.

AddLigandName

```
void AddLigandName(const std::string& ligName)
```

Store an additional ligand name in the *OEResidueCategoryData* associated with this object. Method *UsingExplicitLigandName* will return `true` and components will be recognized as ligands if and only if they have a residue name string matching one of the *OEResidueDatabaseCategory.Ligand* entries.

See also

- *OEMolComplexCategorizer.Constructors*
- *OEMolComplexCategorizer.SetResidueCategoryData*

ClearLigandSubSearch

```
void ClearLigandSubSearch()
```

Clear any ligand subsearch pattern.

See also

- *OEMolComplexCategorizer.GetLigandSubSearch*
- *OEMolComplexCategorizer.SetLigandSubSearch*

CreateCopy

```
OEMolComplexCategorizerBase* CreateCopy() const
```

CreateCopy method.

GetBundleCofactors

```
bool GetBundleCofactors() const
```

Returns `true` if cofactors (other than metals) are associated with the category *OEMolComplexFilterCategory.ProtComplex*.

GetLigandSubSearch

```
OEChem::OESubSearch GetLigandSubSearch() const
```

Return the subsearch pattern used to identify the ligand by graph.

See also

- *Substructure Search* section
- *OEMolComplexCategorizer.SetLigandSubSearch*
- *OEMolComplexCategorizer.ClearLigandSubSearch*

GetMaxLigAtoms

```
unsigned GetMaxLigAtoms() const
```

Return the maximum number of non-hydrogen atoms in a ligand.

See also

- *OEMolComplexCategorizer.SetMaxLigAtoms*

GetMaxLigResidues

```
unsigned GetMaxLigResidues() const
```

See also

- *OEMolComplexCategorizer.SetMaxLigResidues*

Return the maximum number of residues in a ligand.

GetMaxPeptideResidues

```
unsigned GetMaxPeptideResidues() const
```

Return the maximum number of residues in a peptide.

GetMaxProteinAtoms

```
unsigned GetMaxProteinAtoms() const
```

Return the maximum number of non-hydrogen atoms in a protein.

GetMinLigAtoms

```
unsigned GetMinLigAtoms() const
```

Return the minimum number of non-hydrogen atoms in a ligand.

See also

- *OEMolComplexCategorizer.SetMinLigAtoms*

GetResidueCategoryData

```
OEResidueCategoryData* GetResidueCategoryData() const
```

Return a new copy of the *OEResidueCategoryData* object used to define all the molecular components that are *not* ligands.

```
OEResidueCategoryData& GetResidueCategoryData()
```

Returns a reference to the *OEResidueCategoryData* object used to define all the molecular components that are *not* ligands.

SetBundleCofactors

```
void SetBundleCofactors(bool bundle = true)
```

Controls whether cofactors (other than metals) should be associated with the category *OEMolComplexFilterCategory.ProtComplex*.

SetLigandSubSearch

```
void SetLigandSubSearch(const char* smarts)
void SetLigandSubSearch(const OEChem::OESubSearch& ss)
```

Define a subsearch pattern used to identify the ligand by graph.

See also

- *Substructure Search* section
- *OEMolComplexCategorizer.GetLigandSubSearch*
- *OEMolComplexCategorizer.ClearLigandSubSearch*

SetMaxLigAtoms

```
void SetMaxLigAtoms(unsigned n)
```

Set the maximum number of non-hydrogen atoms in a ligand. Default = 70.

Note

When no ligand is found, other molecules such as peptides, nucleic acids, or sugars may be treated as ligands. If this occurs, this restriction will not apply.

SetMaxLigResidues

```
void SetMaxLigResidues(unsigned n)
```

Set the maximum number of residues in a ligand. Default = 3.

Note

When no ligand is found, other molecules such as peptides, nucleic acids, or sugars may be treated as ligands. If this occurs, this restriction will not apply.

SetMaxPeptideResidues

```
void SetMaxPeptideResidues(unsigned n)
```

Set the maximum number of residues to be considered a peptide. Default = 49.

SetMaxProteinAtoms

```
void SetMaxProteinAtoms(unsigned n)
```

Set the maximum number of non-hydrogen atoms in a protein. Default = 0, meaning unlimited.

SetMinLigAtoms

```
void SetMinLigAtoms(unsigned n)
```

Set the minimum number of non-hydrogen atoms in a ligand. Default = 4.

> **Note**
>
> When no ligand is found, other molecules such as peptides, nucleic acids, or sugars may be treated as ligands. If this occurs, this restriction will not apply.

SetResidueCategoryData

```
void SetResidueCategoryData(const OEResidueCategoryData& db)
```

Set the *OEResidueCategoryData* object used to define all the molecular components that are *not* ligands.

UsingExplicitLigandName

```
bool UsingExplicitLigandName() const
```

Returns `true` if an explicit ligand residue name is being used to identify the ligand.

OEMolComplexCategorizerBase

```
class OEMolComplexCategorizerBase
```

Abstract base class functor for objects stored in *OESplitMolComplexOptions* objects which are used by *OESplitMolComplex* and *OEGetMolComplexComponents* to assign functional roles to molecular fragments.

The following *OESplitMolComplexOptions* methods get or set an *OEMolComplexCategorizerBase*:

GetCategorizer	*SetCategorizer*

The following classes derive from this class:

- *OEMolComplexCategorizer*

operator()

```
std::string operator()(OESystem::OERoleSet& roles, OEMolComplexResCounts& counts, const␣
↪OEChem::OEAtomBondSet& frag, bool isCovalent = false) const = 0
```

Method to examine the *OEAtomBondSet* `frag` and return three things required by *OESplitMolComplex* and *OEGetMolComplexComponents* to process the component:

- a new title string
- a *OERoleSet* containing the functional roles associated with this fragment (see *OEMolComplexFilterCategory*)
- an updated map of how many times a residue has been seen to date

If `isCovalent` is true, additional tests may be performed to determine whether the fragment might be considered a (covalent) ligand which is indicated with the *OEMolComplexFilterCategory.Ligand* *OERole* in `roles`.

CreateCopy

```
OEMolComplexCategorizerBase* CreateCopy() const = 0
```

CreateCopy method.

GetBundleCofactors

```
bool GetBundleCofactors() const
```

Returns `true` if cofactors (other than metals) are associated with the category *OEMolComplexFilterCategory.Cofactor*.

GetResidueCategoryData

```
OEResidueCategoryData* GetResidueCategoryData() const = 0
```

Pure virtual method.

```
OEResidueCategoryData& GetResidueCategoryData() = 0
```

Pure virtual method.

UsingExplicitLigandName

```
bool UsingExplicitLigandName() const
```

Returns `true` if an explicit ligand residue name is being used to identify the ligand.

OEMolComplexFilter

```
class OEMolComplexFilter : public OESystem::OEUnaryPredicate<OESystem::OERoleSet>
```

The class *OEMolComplexFilter* represents a predicate over *OERoleSet* objects used by *OESplitMolComplex* and *OEGetMolComplexComponents* to filter components by their functional roles.

The following factories are used to generate *OEMolComplexFilter* objects:

OEMolComplexFilterFactory	*OERoleMolComplexFilterFactory*

OEMolComplexFilter objects can be combined into arbitrarily complex expressions using composition functors (see the *Composite Functors* section).

Constructors

```
OEMolComplexFilter()
OEMolComplexFilter(const OEMolComplexFilter& rhs)
```

Constructor and copy constructor.

operator()

```
bool operator()(const OESystem::OERoleSet& roles) const
```

Returns `true` if the *OERoleSet* matches the filter.

CreateCopy

```
OESystem::OEUnaryFunction<OESystem::OERoleSet, bool>* CreateCopy() const
```

CreateCopy method.

OEPerceiveInteractionOptions

```
class OEPerceiveInteractionOptions
```

This class is a container for all the parameters that control interaction perception when invoking the *OEPerceiveInteractionHints* function.

The current list of interactions include:

- **cation-pi** (see also *OECationPiInteractionHint* class and *OECationPiInteractionHintType* namespace)
- **clashes** (see also *OEClashInteractionHint* class)
- **contacts** (see also *OEContactInteractionHint* class)
- **metal interactions** (see also *OEChelatorInteractionHint* class and *OEChelatorInteractionHintType* namespace)
- **covalents** (see also *OECovalentInteractionHint* class)
- **halogen bonds** (see also *OEHalogenBondInteractionHint* class and *OEHalogenBondInteractionHintType* namespace)
- **hydrogen bonds** (see also *OEHBondInteractionHint* class and *OEHBondInteractionHintType* namespace)
- **salt-bridges** (see also *OESaltBridgeInteractionHint* class and *OESaltBridgeInteractionHintType* namespace)
- **Pi and T stacking** (see also *OEStackingInteractionHint* class and *OEStackingInteractionHintType* namespace)

Each interaction type has one or more parameters that control its perception. In addition, hydrogen-bonding clashes, intramolecular hydrogen-bonds, unpaired hydrogen-bonds and unpaired salt-bridges are also perceived. Unpaired interactions are those that are near the protein-ligand interface, but do not form an apparent bond.

See also

- [Kumar-2002]

- [Cavallo-2016]
- [Bissantz-2010]
- [Marcou-2007]

Get method	Set method	Interaction type(s)	Default value
GetExtendedSaltBridge		salt bridge	
GetMaxAcceptorAngle	SetMaxAcceptorAngle	hbond	30 degrees
GetMaxAtomPairDistanc			
GetMaxCationPiAngle	SetMaxCationPiAngle	cation-Pi	50 degrees
GetMaxCationPiDistanc	SetMaxCationPiDistanc	cation-Pi	5.5 Ångströms
GetMaxChargeAidedHBon	SetMaxChargeAidedHBon	hbond	3.5 Ångströms
GetMaxContactFraction	SetMaxContactFraction	contact	1.2
GetMaxCovalentFractio	SetMaxCovalentFractio	covalent	1.1
GetMaxDonorAngle	SetMaxDonorAngle	hbond	35 degrees
GetMaxHBondDistance	SetMaxHBondDistance	hbond	3.2 Ångströms
GetMaxHalogenAngle	SetMaxHalogenAngle	halogen bond	35 degrees
GetMaxHalogenBondDist	SetMaxHalogenBondDist	halogen bond	3.2 Ångströms
GetMaxMetalBondDistan	SetMaxMetalBondDistan	chelator	3.5 Ångströms
GetMaxNonIdealAccepto	SetMaxNonIdealAccepto	hbond	30 degrees
GetMaxNonIdealDonorAn	SetMaxNonIdealDonorAn	hbond	45 degrees
GetMaxNonIdealHBondDi	SetMaxNonIdealHBondDi	hbond	3.8 Ångströms
GetMaxPiStackAngle	SetMaxPiStackAngle	Pi-stacking	$\frac{\pi}{9.0}$
GetMaxPiStackDistance	SetMaxPiStackDistance	Pi-stacking	5.0 Ångströms
GetMaxProteinIntraDis	SetMaxProteinIntraDis	intramolecular (protein)	6.0 Ångströms
GetMaxSaltBridgeDista	SetMaxSaltBridgeDista	salt bridge	5.0 Ångströms
GetMaxTStackAngle	SetMaxTStackAngle	T-stacking	$\frac{\pi}{9.0}$
GetMaxTStackDistance	SetMaxTStackDistance	T-stacking	5.35 Ångströms
GetMaxUnpairedAngle	SetMaxUnpairedAngle	hbond	$\frac{\pi}{2}$
GetMaxUnpairedDistanc	SetMaxUnpairedDistanc	hbond	5.0 Ångströms
GetMinContactFraction	SetMinContactFraction	contact, clash	0.8
GetMinCovalentFractio	SetMinCovalentFractio	covalent	0.9
GetMinHBondContactFra	SetMinHBondContactFra	hbond	0.5
GetMinStackOffset	SetMinStackOffset	Pi-stacking, T-stacking	0.5 Ångströms

Constructors

```
OEPerceiveInteractionOptions()
```

The default constructor for perception of interaction options. It sets all of the interaction parameters to their default value.

GetExtendedSaltBridgeDistance

```
double GetExtendedSaltBridgeDistance() const
```

Actual salt bridge interaction distance is calculated from the geometric centers of the two interacting charge groups, which may or may not be comprised of a single atom. The distance returned by this function is the maximum distance between two formally charged atoms for subsequent identification of charge groups. This is a simple increment larger than the value associated with *OEPerceiveInteractionOptions.GetMaxSaltBridgeDistance* and

OEPerceiveInteractionOptions.SetMaxSaltBridgeDistance.

GetMaxAcceptorAngle

```
double GetMaxAcceptorAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxAcceptorAngle*.

GetMaxAtomPairDistance

```
double GetMaxAtomPairDistance() const
```

Returns the maximum value of

- *OEPerceiveInteractionOptions.GetMaxHBondDistance*,
- *OEPerceiveInteractionOptions.GetMaxChargeAidedHBondDistance*,
- *OEPerceiveInteractionOptions.GetMaxMetalBondDistance*,
- *OEPerceiveInteractionOptions.GetExtendedSaltBridgeDistance*, and
- *OEPerceiveInteractionOptions.GetMaxHalogenBondDistance*.

GetMaxCationPiAngle

```
double GetMaxCationPiAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxCationPiAngle*.

GetMaxCationPiDistance

```
double GetMaxCationPiDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxCationPiDistance*.

GetMaxChargeAidedHBondDistance

```
double GetMaxChargeAidedHBondDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxChargeAidedHBondDistance*.

GetMaxContactFraction

```
double GetMaxContactFraction() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxContactFraction*.

GetMaxCovalentFraction

```
double GetMaxCovalentFraction() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxCovalentFraction*.

GetMaxDonorAngle

```
double GetMaxDonorAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxDonorAngle*.

GetMaxHBondDistance

```
double GetMaxHBondDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxHBondDistance*.

GetMaxHalogenAngle

```
double GetMaxHalogenAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxHalogenAngle*.

GetMaxHalogenBondDistance

```
double GetMaxHalogenBondDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxHalogenBondDistance*.

GetMaxMetalBondDistance

```
double GetMaxMetalBondDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxMetalBondDistance*.

GetMaxNonIdealAcceptorAngle

```
double GetMaxNonIdealAcceptorAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxNonIdealAcceptorAngle*.

GetMaxNonIdealDonorAngle

```
double GetMaxNonIdealDonorAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxNonIdealDonorAngle*.

GetMaxNonIdealHBondDistance

```
double GetMaxNonIdealHBondDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxNonIdealHBondDistance*.

GetMaxPiStackAngle

```
double GetMaxPiStackAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxPiStackAngle*.

GetMaxPiStackDistance

```
double GetMaxPiStackDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxPiStackDistance*.

GetMaxProteinIntraDistance

```
double GetMaxProteinIntraDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxProteinIntraDistance*.

GetMaxSaltBridgeDistance

```
double GetMaxSaltBridgeDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxSaltBridgeDistance*.

GetMaxTStackAngle

```
double GetMaxTStackAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxTStackAngle*.

GetMaxTStackDistance

```
double GetMaxTStackDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxTStackDistance*.

GetMaxUnpairedAngle

```
double GetMaxUnpairedAngle() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxUnpairedAngle*.

GetMaxUnpairedDistance

```
double GetMaxUnpairedDistance() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMaxUnpairedDistance*.

GetMinContactFraction

```
double GetMinContactFraction() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMinContactFraction*.

GetMinCovalentFraction

```
double GetMinCovalentFraction() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMinCovalentFraction*.

GetMinHBondContactFraction

```
double GetMinHBondContactFraction() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMinHBondContactFraction*.

GetMinStackOffset

```
double GetMinStackOffset() const
```

Returns the value set by *OEPerceiveInteractionOptions.SetMinStackOffset*.

SetMaxAcceptorAngle

```
void SetMaxAcceptorAngle(double radians)
```

In perception of a hydrogen bond, we define the angle $X-A...D$, where **A** and **D** are the acceptor and donor respectively, and **X** is one or more points in space that are located so that an idealized hydrogen-bond would give an angle of π. The maximum acceptor angle is the maximum allowed deviation (in radians) from the idealized angle (or the minimum deviation from multiple angles when more than one **X** position is possible).

Default= $(\pi * \frac{6.0}{36.0})$ **= 30 degrees**

See also

- *OEPerceiveInteractionOptions.SetMinHBondContactFraction* method
- *OEPerceiveInteractionOptions.SetMaxHBondDistance* method
- *OEPerceiveInteractionOptions.SetMaxDonorAngle* method
- *OEPerceiveInteractionOptions.SetMaxChargeAidedHBondDistance* method

SetMaxCationPiAngle

```
void SetMaxCationPiAngle(double radians)
```

This option sets the maximum allowed angle defined by a point on the vector normal from the pi ring system, the geometric center of the ring system and the geometric center of the charged system. For the purposes of Cation Pi interaction hints, only Lysine, N-methyl Lysine and Arginine are recognized cations. For Lysine and N-methyl Lysine, the charge center is the charged Nitrogen atom, while for the Arginine residue, the charge center will be the geometric

center of the Carbon and three Nitrogen atoms that share the resonant charge. Each smallest aromatic ring (independent of ring fusions) is considered separately and a cation can interact with more than one pi system simultaneously.

Default= $(\pi * \frac{10.0}{36.0})$ **= 50 degrees**

See also

- *OEPerceiveInteractionOptions.SetMaxCationPiDistance* method

SetMaxCationPiDistance

```
void SetMaxCationPiDistance(double dist)
```

This is the maximum allowed distance between the center of a cation system and the center of an aromatic pi system. For the purposes of Cation Pi interaction hints, only Lysine, N-methyl Lysine and Arginine are recognized cations. Each smallest aromatic ring (independent of ring fusions) is considered separately and a cation can interact with more than one pi system simultaneously.

Default= 5.5 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMaxCationPiAngle* method

SetMaxChargeAidedHBondDistance

```
void SetMaxChargeAidedHBondDistance(double dist)
```

For the purpose of hydrogen bond perception, charge assisted hydrogen bonds, are identified when one of the two bonding partners is part of a functional group with a net formal charge. These interactions include dipole - charge interactions and are perceived at slightly longer range than simple hydrogen- bonds by default.

Default= 3.5 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMinHBondContactFraction* method
- *OEPerceiveInteractionOptions.SetMaxHBondDistance* method
- *OEPerceiveInteractionOptions.SetMaxAcceptorAngle* method
- *OEPerceiveInteractionOptions.SetMaxDonorAngle* method

SetMaxContactFraction

```
void SetMaxContactFraction(double frac)
```

Van der Waals interactions are perceived as non-interacting, in attractive contact, or clashing. The three classes are determined by comparing the atom - atom distance **Daa** to the sum of the van der Waals radii of the two atoms **VDWtot**. The distance **Daa** is represented as a fraction of **VDWtot** which we will call **Faa**. If the value of **Faa** is greater than the value stored in *OEPerceiveInteractionOptions.SetMaxContactFraction*, then the two

atoms are non-interacting. If the value of **Faa** is between the values set by *OEPerceiveInteractionOptions.SetMaxContactFraction* and *OEPerceiveInteractionOptions.SetMinContactFraction* then the atoms are in attractive contact. If the value of **Faa** is less than *OEPerceiveInteractionOptions.SetMinContactFraction* then the atoms are clashing.

Any two atoms that are forming a hydrogen bond can be in attractive contact at shorter distances than generic atom pairs because of the balance of forces. To handle this, when two atoms are forming a hydrogen bond, the value of **Faa** is compared to *OEPerceiveInteractionOptions.SetMinHBondContactFraction* to determine clashing rather than *OEPerceiveInteractionOptions.SetMinContactFraction*.

Default = 1.2

See also

- *OEPerceiveInteractionOptions.SetMinContactFraction* method
- *OEPerceiveInteractionOptions.SetMinHBondContactFraction* method

SetMaxCovalentFraction

```
void SetMaxCovalentFraction(double frac)
```

This function sets the maximum ratio of the distance between to atoms to the sum of their covalent radii allowed for perception of a covalent bond.

Covalent interactions are perceived based on comparing the distance between two atoms **D** to the sum of the covalent radii **SumRc**. If **D** is > **SumRc** times the value set by *OEPerceiveInteractionOptions.SetMinCovalentFraction* and **D** is < **SumRc** times the value set by *OEPerceiveInteractionOptions.SetMaxCovalentFraction*, then a covalent interaction will be perceived. Currently no angle constraint is used in perceiving covalent interactions.

Hydrogen and Halide atoms as well as water molecules are excluded from covalent interaction hint perception. Metal atom interactions are perceived as *OEChelatorInteractionHint* rather than as covalent interactions.

When a covalent interaction exists between atom **P** of the protein and atom **L** of the ligand, then no other interactions will be perceived for atoms **P** and **L**. Further, atoms bonded to atoms **L** and **P** will be excluded from perception of clash or contact interactions. Bonded atoms will be able to participate in other interactions such as hydrogen bonding.

Default = 1.1

See also

- *OEPerceiveInteractionOptions.SetMinCovalentFraction* method

SetMaxDonorAngle

```
void SetMaxDonorAngle(double radians)
```

In the perception of a hydrogen bond with the interaction $A...H-D$, the angle $A-D-H$ is ideally zero in most cases. This function determines the maximum allowed value of this angle in a perceived hydrogen-bond.

Default = $(\pi * \frac{7.0}{36.0})$ **= 35 degrees**

See also

- *OEPerceiveInteractionOptions.SetMinHBondContactFraction* method
- *OEPerceiveInteractionOptions.SetMaxHBondDistance* method
- *OEPerceiveInteractionOptions.SetMaxAcceptorAngle* method
- *OEPerceiveInteractionOptions.SetMaxChargeAidedHBondDistance* method

SetMaxHBondDistance

```
void SetMaxHBondDistance(double dist)
```

This function sets the maximum distance between the donor and acceptor heavy atoms for a hydrogen-bond to be perceived.

Default = 3.2 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMinHBondContactFraction* method
- *OEPerceiveInteractionOptions.SetMaxAcceptorAngle* method
- *OEPerceiveInteractionOptions.SetMaxDonorAngle* method
- *OEPerceiveInteractionOptions.SetMaxChargeAidedHBondDistance* method

SetMaxHalogenAngle

```
void SetMaxHalogenAngle(double radians)
```

In the perception of halogen bonds, there is an angle formed between $R - X...E/N$ where **X** is a univalent halogen, **E/N** are the electrophile or nucleophile, and **R** is the attachment of the halogen. For nucleophiles, the ideal interaction angle is π, for electrophiles, the ideal interaction angle $\frac{\pi}{2}$. This function determines the maximum allowed deviation (in radian) from these ideal angles, respectively, for perception of a halogen bond.

Default = $(\pi * \frac{7.0}{36.0})$ **= 35 degrees**

See also

- *OEPerceiveInteractionOptions.SetMaxHalogenBondDistance* method

SetMaxHalogenBondDistance

```
void SetMaxHalogenBondDistance(double dist)
```

This is the maximum distance between the halogen atom and the electrophile or nucleophile atom for perception of a halogen bond.

Default = 3.2 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMaxHalogenAngle* method

SetMaxMetalBondDistance

```
void SetMaxMetalBondDistance(double dist)
```

This is the maximum distance allowed for an interaction to be perceived between a metal atom and a chelating non-metal atom.

Default = 3.5 Ångströms

SetMaxNonIdealAcceptorAngle

```
void SetMaxNonIdealAcceptorAngle(double radians)
```

Sets the maximum angle deviation (in radian) from the non-ideal angle.

Default= $(\pi * \frac{6.0}{36.0})$ **= 30 degrees**

See also

- *OEPerceiveInteractionOptions.SetMaxNonIdealHBondDistance* method

SetMaxNonIdealDonorAngle

```
void SetMaxNonIdealDonorAngle(double radians)
```

Sets the maximum allowed value of this angle in a perceived non-ideal hydrogen bond.

Default= $(\pi * \frac{9.0}{36.0})$ **= 45 degrees**

See also

- *OEPerceiveInteractionOptions.SetMaxNonIdealHBondDistance* method

SetMaxNonIdealHBondDistance

```
void SetMaxNonIdealHBondDistance(double dist)
```

This function sets the maximum distance between the donor and acceptor heavy atoms for a non-ideal hydrogen bond to be perceived.

Default = 3.8 Ångströms

SetMaxPiStackAngle

```
void SetMaxPiStackAngle(double radians)
```

Sets the maximum angle deviation (in radian) from the ideal angle calculated between the two aromatic ring normals to be considered for Pi-stacking interaction.

Default = $\frac{\pi}{9.0}$

See also

- *OEPerceiveInteractionOptions.SetMaxPiStackDistance* method

SetMaxPiStackDistance

```
void SetMaxPiStackDistance(double dist)
```

Sets the maximum distance (in Ångström) allowed between the geometric center of two aromatic rings in order to being considered for Pi-stacking interaction.

Default = 5.0 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMaxPiStackAngle* method

SetMaxProteinIntraDistance

```
void SetMaxProteinIntraDistance(double dist)
```

Sets the maximum distance (in Ångströms) used to identify protein atoms that are neighbours of the ligand when perceiving intramolecular protein hydrogen bonds and chelator interactions.

Default = 6.0 Ångströms

SetMaxSaltBridgeDistance

```
void SetMaxSaltBridgeDistance(double dist)
```

Sets the maximum distance (in Ångströms) allowed between the geometric centers of the two formally charged groups in a perceived salt bridge interaction.

Default = 5.0 Ångströms

See also

- *OEPerceiveInteractionOptions.GetExtendedSaltBridgeDistance* method

SetMaxTStackAngle

```
void SetMaxTStackAngle(double radians)
```

Sets the maximum angle deviation (in radian) from the ideal angle ($\frac{\pi}{2}$) calculated between the two aromatic ring normals to be considered for T-stacking interaction.

Default = $\frac{\pi}{9.0}$

See also

- *OEPerceiveInteractionOptions.SetMaxTStackDistance* method

SetMaxTStackDistance

```
void SetMaxTStackDistance(double dist)
```

Sets the maximum distance (in Ångström) allowed between the geometric center of two aromatic rings in order to being considered for T-stacking interaction.

Default = 5.35 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMaxTStackAngle* method

SetMaxUnpairedAngle

```
void SetMaxUnpairedAngle(double radians)
```

In perception of unpaired interactions, the unpaired potential interaction group should be oriented toward the ligand or protein where it could make an interaction. This is the maximum deviation (in radian) allowed between the potential interaction group and the nearest potential partner atom.

Default = $\frac{\pi}{2}$

See also

- *OEPerceiveInteractionOptions.SetMaxUnpairedDistance* method

SetMaxUnpairedDistance

```
void SetMaxUnpairedDistance(double dist)
```

In perception of unpaired interactions, the unpaired potential interaction group should be within a minimum distance of the nearest potential paired atom. This function sets the maximum distance (in Ångströms) for perception of an unpaired potential.

Default = 5.0 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMaxUnpairedAngle* method

SetMinContactFraction

```
void SetMinContactFraction(double frac)
```

der Waals interactions are perceived as non-interacting, in attractive contact, or clashing. The three classes are determined by comparing the atom - atom distance **Daa** to the sum of the van der Waals radii of the two atoms **VDWtot**. The distance **Daa** is represented as a fraction of **VDWtot** which we will call **Faa**. If the value of **Faa** is greater than the value stored in *OEPerceiveInteractionOptions.SetMaxContactFraction*, then the two atoms are non-interacting. If the value of **Faa** is between the values set by *OEPerceiveInteractionOptions.SetMaxContactFraction* and *OEPerceiveInteractionOptions.SetMinContactFraction* then the atoms are in attractive contact. If the value of **Faa** is less than *OEPerceiveInteractionOptions.SetMinContactFraction* then the atoms are clashing.

Any two atoms that are forming a hydrogen bond can be in attractive contact at shorter distances than generic atom pairs because of the balance of forces. To handle this, when two atoms are forming a hydrogen bond, the value of **Faa** is compared to *OEPerceiveInteractionOptions.SetMinHBondContactFraction* to determine clashing rather than *OEPerceiveInteractionOptions.SetMinContactFraction*.

Default = 0.8

See also

- *OEPerceiveInteractionOptions.SetMaxContactFraction* method
- *OEPerceiveInteractionOptions.SetMinHBondContactFraction* method

SetMinCovalentFraction

```
void SetMinCovalentFraction(double frac)
```

This function sets the minimum ratio of the distance between to atoms to the sum of their covalent radii allowed for perception of a covalent bond.

Covalent interactions are perceived based on comparing the distance between two atoms **D** to the sum of the covalent radii **SumRc**. If **D** is > **SumRc** times the value set by *OEPerceiveInteractionOptions.SetMinCovalentFraction* and **D** is < **SumRc** times the value set by *OEPerceiveInteractionOptions.SetMaxCovalentFraction*, then a covalent interaction will be perceived. Currently no angle constraint is used in perceiving covalent interactions.

Hydrogen and Halide atoms as well as water molecules are excluded from covalent interaction hint perception. Metal atom interactions are perceived as *OEChelatorInteractionHint* rather than as covalent interactions.

When a covalent interaction exists between atom **P** of the protein and atom **L** of the ligand, then no other interactions will be perceived for atoms **P** and **L**. Further, atoms bonded to atoms **L** and **P** will be excluded from perception of clash or contact interactions. Bonded atoms will be able to participate in other interactions such as hydrogen bonding.

Default = 0.9

See also

- *OEPerceiveInteractionOptions.SetMaxCovalentFraction* method

SetMinHBondContactFraction

```
void SetMinHBondContactFraction(double frac)
```

der Waals interactions are perceived as non-interacting, in attractive contact, or clashing. The three classes are determined by comparing the atom - atom distance **Daa** to the sum of the van der Waals radii of the two atoms **VDWtot**. The distance **Daa** is represented as a fraction of **VDWtot** which we will call **Faa**. If the value of **Faa** is greater than the value stored in *OEPerceiveInteractionOptions.SetMaxContactFraction*, then the two atoms are non-interacting. If the value of **Faa** is between the values set by *OEPerceiveInteractionOptions.SetMaxContactFraction* and *OEPerceiveInteractionOptions.SetMinContactFraction* then the atoms are in attractive contact. If the value of **Faa** is less than *OEPerceiveInteractionOptions.SetMinContactFraction* then the atoms are clashing.

Any two atoms that are forming a hydrogen bond can be in attractive contact at shorter distances than generic atom pairs because of the balance of forces. To handle this, when two atoms are forming a hydrogen bond, the value of **Faa** is compared to *OEPerceiveInteractionOptions.SetMinHBondContactFraction* to determine clashing rather than *OEPerceiveInteractionOptions.SetMinContactFraction*.

Default = 0.5

See also

- *OEPerceiveInteractionOptions.SetMaxContactFraction* method
- *OEPerceiveInteractionOptions.SetMinContactFraction* method
- *OEPerceiveInteractionOptions.SetMinHBondContactFraction* method
- *OEPerceiveInteractionOptions.SetMaxHBondDistance* method
- *OEPerceiveInteractionOptions.SetMaxAcceptorAngle* method
- *OEPerceiveInteractionOptions.SetMaxDonorAngle* method
- *OEPerceiveInteractionOptions.SetMaxChargeAidedHBondDistance* method

SetMinStackOffset

```
void SetMinStackOffset(double dist)
```

Sets the minimum allowed lateral offset (in Ångströms) used to filter Pi-stacking and T-stacking interaction candidates.

Default = 0.5 Ångströms

See also

- *OEPerceiveInteractionOptions.SetMaxPiStackDistance* method
- *OEPerceiveInteractionOptions.SetMaxTStackDistance* method

OEPlaceHydrogensDetails

```
class OEPlaceHydrogensDetails
```

The *read-only* container *OEPlaceHydrogensDetails* holds information about the results of applying the function *OEPlaceHydrogens* to a particular molecule. Details objects can provide insight into the conformational states and how they were chosen.

Information is broken down into clusters (including singletons) of "movers" or moving functional groups and their associated "background" of non-moving groups. Information about each mover is collected in a corresponding *OEPlaceHydrogensMoverNote*.

See also

- *Adding Hydrogens and Optimizing Hydrogen Bonds* section
- *OEPlaceHydrogens* function
- *OEPlaceHydrogensMoverNote* class

Constructors

```
OEPlaceHydrogensDetails()
OEPlaceHydrogensDetails(const OEPlaceHydrogensDetails& rhs)
```

Default and copy constructor.

ClusterHasBadClash

```
bool ClusterHasBadClash(unsigned clusterIdx) const
```

Returns `true` if the cluster referred to by *clusterIdx* is in a conformation that generates a "bad" atom-atom clash.

See also

- *GetNumClusters* method
- *OEPlaceHydrogensOptions.SetBadClashOverlapDistance* method
- *OEPlaceHydrogensOptions.GetBadClashOverlapDistance* method

Describe

```
std::string Describe(bool showBackground = false) const
```

Returns a multi-line description of the conformation and score of movers in each cluster processed by the *OEPlaceHydrogens* function.

```
OEBio 2.1.1: score system MMFF-NIE: flip bias 2.000
87 clusters : 5 flips
cluster 0  : score -53.935!
             CG  ASN44(A)   :      amide:   -6.084  (o=-53.93!,f=-8.42!)
```

(continues on next page)

(continued from previous page)

```
             NZ  LYS47(A)   : NH3 150deg:  -14.865
             CG  HIS80(A)   :    +bothHN:   -5.628  (o=-53.93!,f=-15.89!)
             OG1 THR81(A)   : OH   65deg:    1.255
             CD  GLN87(A)   :FLIP  amide:   -4.976  (o=-26.48!,f=-53.93!)
             CG  ASN126(A)  :      amide:   -5.193  (o=-53.93!,f=-35.33!)
             CG  ASN169(A)  :      amide:   -2.230  (o=-53.93!,f=-33.02!)
             CD  GLN203(A)  :      amide:   -1.400  (o=-53.93!,f=-15.53!)
             CG  HIS205(A)  :    +bothHN:   -8.693! (o=-53.93!,f=-6.89!)
             OG1 THR232(A)  : OH  179deg:    9.120
             O3B XYP601(A)  : OH  280deg:   -5.020
             O2B XYP601(A)  : OH  173deg:    3.917
             O4B XYP601(A)  : OH   50deg:    1.236
             O3  XIF602(A)  : OH   29deg:   -3.688
cluster 1  : score -85.724!
             CG  HIS85(A)   :    +bothHN:   -7.737! (o=-85.72!,f=-84.85!)
             OG  SER86(A)   : OH  241deg:    2.116
             OG  SER139(A)  : OH  185deg:    3.367!
             OH  TYR171(A)  : OH   25deg:   -3.384
             CG  ASN172(A)  :      amide:    1.153  (o=-85.72!,f=-68.75!)
             NZ  LYS179(A)  : NH3 188deg:  -39.497
...
single  13 : OG1 THR2(A)    : OH  151deg:    1.545
single  14 : OG  SER25(A)   : OH  292deg:    2.046
single  15 : OH  TYR29(A)   : OH  199deg:   -1.262
single  16 : OG  SER35(A)   : OH  254deg:    1.935
...
```

Key elements in the description output:

- The score system is `MMFF-NIE` which stands for "MMFF Neighbor Interaction Energies". This is is a physics based score where the Coulombic and Van der Waals terms are from MMFF94 (including MMFF partial charges), with a weighting factor of 1:3 Coulomb to VDW. There are biases against ionization and "flipping" a group. Interactions are scored between movers and between movers and non-moving background atoms. Background atoms are neighboring atoms that form a "dipole cluster", the sum of the partial charges of these clusters are the same as the sum of the formal charges.
- The count `87 clusters` includes both groups of interacting movers and singletons, movers that interact only with non-moving atoms or waters.
- `5 flips` indicates that five movers were "flipped" by 180 degrees to improve hydrogen bonding or avoid clashes. Groups such as the asparagine sidechain `CD GLN87(A) :FLIP amide:` are occasionally modeled with certain heavy atoms in the wrong orientation. Any free amide or imidazole group will be considered for flipping.
- The score in `OG SER139(A) : OH 185deg:  3.367!` is from `score system MMFF-NIE` for this mover in the specified conformation. Lower numbers represent better scores. The exclamation point (!) indicates a clash between this group and another group or non-moving background atom.
- The score in `cluster 0 :  score -53.935!` is the sum of both the mover scores, and the background scores (not shown here, but displayable if the `showBackground` parameter is set to `true`). The exclamation point (!) indicates that at least one mover had a clash.
- The pair of scores for flippable groups `(o=-53.93!,f=-8.42!)` are the best cluster scores for both the original (o) and the flipped (f) state. After the basic optimization, each flippable mover in a cluster is fixed in the opposite flip state and the optimization is re-run. A group is flipped only if the flipped cluster score is lower than the original score by more than `flip bias 2.000`.

- Imidazoles have both flipped states and three protonation states for each flip state. The description indicates whether the group is flipped and where protons are with respect to the two ring nitrogens: `: no HE2:` *neutral with HD1 but no HE2* or `:FLIP no HD1:` *flipped neutral with HE2 but no HD1* or `: +bothHN: +` *charged with two NH protons.*
- Other groups have descriptions of their state. `O3B XYP601(A) : OH 280deg:` shows the orientation of one of the hydroxyl rotors in the residue XYP.
- Waters (not shown above) are processed after other movers, progressively in shells away from the protein.

See also

- *OEPlaceHydrogensMoverNote.Describe* method
- *OEPlaceHydrogensMoverClass* constants

GetClusterScore

```
double GetClusterScore(unsigned clusterIdx) const
```

Returns the combined score of all the movers (plus non-moving background) of the cluster referred to by *clusterIdx*.

See also

- *GetNumClusters* method

GetFlipBias

```
double GetFlipBias() const
```

Returns the score bias against flipping functional groups (e.g. imidazole, amide) used by *OEPlaceHydrogens* when this details object was produced.

See also

- *OEPlaceHydrogensOptions.GetFlipBias* method

GetMoverNotes

```
OESystem::OEIterBase<OEPlaceHydrogensMoverNote>* GetMoverNotes(unsigned clusterIdx) const
```

Returns an iterator of all the *mover notes* describing the state of movers in the cluster referred to by *clusterIdx*.

See also

- *OEPlaceHydrogensMoverNote*
- *OEIter* class

GetNumClusters

```
unsigned GetNumClusters() const
```

Returns the number of mover clusters processed by *OEPlaceHydrogens* when this details object was produced.

GetNumFlips

```
unsigned GetNumFlips() const
```

Returns the number movers *OEPlaceHydrogens* decided to flip when this details object was produced.

See also

- *OEPlaceHydrogensOptions.GetFlipBias* method

GetNumHydrogensStandardized

```
unsigned GetNumHydrogensStandardized() const
```

Returns the number of bonds between a hydrogen and a heavy atom whose length was adjusted by *OEPlaceHydrogens* when this details object was produced.

See also

- *OEPlaceHydrogensOptions.SetStandardizeBondLen* method

GetNumMovers

```
unsigned GetNumMovers(unsigned clusterIdx) const
```

Returns the number of movers in the cluster referred to by *clusterIdx* (including all non-moving background "movers").

See also

- *GetNumClusters* method
- *IsSingleton* method
- *GetMoverNotes* method

IsSingleton

```
bool IsSingleton(unsigned clusterIdx) const
```

Returns `true` if the cluster referred to by *clusterIdx* only contains a single mover (not counting the corresponding non-moving background).

See also

- *GetNumMovers* method

IsValid

```
bool IsValid() const
```

Returns `true` if this details object contains valid information. Returns `false` if *OEPlaceHydrogens* failed.

OEPlaceHydrogensMoverNote

```
class OEPlaceHydrogensMoverNote
```

The *read-only* container *OEPlaceHydrogensMoverNote* holds information about a single functional group processed by *OEPlaceHydrogens*.

These mover notes are organized into clusters in *OEPlaceHydrogensDetails* objects.

See also

- *Adding Hydrogens and Optimizing Hydrogen Bonds* section
- *OEPlaceHydrogens* function
- *OEPlaceHydrogensDetails* class

Constructors

```
OEPlaceHydrogensMoverNote()
OEPlaceHydrogensMoverNote(const OEPlaceHydrogensMoverNote& rhs)
```

Default and copy constructor.

Describe

```
std::string Describe() const
```

Returns a description of the identity, state, and score of this mover.

See also

- *OEPlaceHydrogensDetails.Describe* method

GetAngle

```
double GetAngle() const
```

Returns the angle this mover was oriented in (if rotatable).

GetFlippedBestClusterScore

```
double GetFlippedBestClusterScore() const
```

Returns the best score for this cluster when this mover was flipped (if flippable).

See also

- *IsFlippable* method
- *IsFlipped* method

GetLabel

```
std::string GetLabel() const
```

Returns a string describing this mover.

GetMoverClass

```
unsigned GetMoverClass() const
```

Returns the *class* of this mover.

See also

- *OEPlaceHydrogensMoverClass* namespace
- *OEGetPlaceHydrogensMoverClassName*

GetOriginalBestClusterScore

```
double GetOriginalBestClusterScore() const
```

Returns the best score for this cluster when this mover was *not* flipped (if flippable).

See also

- *IsFlippable* method
- *IsFlipped* method

GetReferenceCoords

```
bool GetReferenceCoords(double* xyz) const
```

Returns 3-D coordinates of an atom representing the location of this mover.

GetScore

```
double GetScore() const
```

Returns the score for this mover.

GetStateIdx

```
unsigned GetStateIdx() const
```

Returns the index of the conformational state of this mover.

GetStateName

```
std::string GetStateName() const
```

Returns the name of the state of this mover.

HasBadClash

```
bool HasBadClash() const
```

Returns `true` if this mover in a state that makes a bad clash.

See also

- *OEPlaceHydrogensOptions.SetBadClashOverlapDistance* method
- *OEPlaceHydrogensOptions.GetBadClashOverlapDistance* method

HasFlippedBadClusterClash

```
bool HasFlippedBadClusterClash() const
```

Returns `true` if this cluster has a bad clash when this mover is flipped (if flippable).

See also

- *OEPlaceHydrogensOptions.SetBadClashOverlapDistance* method
- *OEPlaceHydrogensOptions.GetBadClashOverlapDistance* method
- *IsFlippable* method
- *IsFlipped* method

HasOriginalBadClusterClash

```
bool HasOriginalBadClusterClash() const
```

Returns `true` if this cluster has a bad clash when this mover is *not* flipped (if flippable).

See also

- *OEPlaceHydrogensOptions.SetBadClashOverlapDistance* method
- *OEPlaceHydrogensOptions.GetBadClashOverlapDistance* method
- *IsFlippable* method
- *IsFlipped* method

IsFlippable

```
bool IsFlippable() const
```

Returns `true` if this this mover is flippable.

See also

- *IsFlipped* method

IsFlipped

```
bool IsFlipped() const
```

Returns `true` if this mover has been flipped. If restrained, it may be flipped even though it is not flippable.

See also

- *IsFlippable* method
- *IsRestrained* method
- *GetFlippedBestClusterScore* method
- *GetOriginalBestClusterScore* method
- *HasFlippedBadClusterClash* method
- *HasOriginalBadClusterClash* method
- *OEPlaceHydrogensOptions.SetFlipPredicate* method

IsIonized

```
bool IsIonized() const
```

Returns `true` if this mover is in an ionized state.

IsIonizable

```
bool IsIonizable() const
```

Returns `true` if this mover is ionized in one of its allowed states.

IsRestrained

```
bool IsRestrained() const
```

Returns `true` if this mover is restrained as to the allowed states. For example, if it is restrained to be flipped.

See also

- *IsFlipped* method
- *OEPlaceHydrogensOptions.SetFlipPredicate* method
- *OEPlaceHydrogensOptions.SetNoFlipPredicate* method

OEPlaceHydrogensOptions

```
class OEPlaceHydrogensOptions : public OESystem::OEOptions
```

The *OEPlaceHydrogensOptions* class encapsulates properties and resources that determine how *OEPlaceHydrogens* functions.

See also

- *Adding Hydrogens and Optimizing Hydrogen Bonds* section
- *OEPlaceHydrogens* function

Constructors

```
OEPlaceHydrogensOptions(bool compatibleAlts = false)
OEPlaceHydrogensOptions(const OEPlaceHydrogensOptions& rhs)
```

Constructor and copy constructor. The optional parameter `compatibleAlts` controls how alternate location codes are interpreted by *OEPlaceHydrogens*.

See also

- *SetAltsMustBeCompatible* method
- *GetAltsMustBeCompatible* method

GetAddMetadata

```
bool GetAddMetadata() const
```

Returns `true` if metadata describing the results will be added to the molecule. The metadata is the string generated by the *OEPlaceHydrogensDetails.Describe* method and can be accessed by the *OEGetPlaceHydrogensMetadata* function.

See also

- *OEHasPlaceHydrogensMetadata* function
- *OEGetPlaceHydrogensMetadata* function
- *OEClearPlaceHydrogensMetadata* function
- *OEPlaceHydrogensDetails* class

GetAltsMustBeCompatible

```
bool GetAltsMustBeCompatible() const
```

Returns `true` if alternate location codes are required to be compatible (either identical or blank) when atoms interact. If `true`, incompatible atoms do not interact.

See also

- *Constructors*

GetBadClashOverlapDistance

```
double GetBadClashOverlapDistance() const
```

Returns the amount of excess van der Waals overlap considered to be a "bad" clash.

GetBypassPredicate

```
const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>* GetBypassPredicate() const
```

Returns a new copy of the predicate that *OEPlaceHydrogens* will use to exclude from analysis all movers having a matching atom.

(Default: *OEIsFalseAtom*)

See also

- *SetBypassPredicate* method
- *GetFlipPredicate* method
- *GetNoFlipPredicate* method

GetFlipBias

```
double GetFlipBias() const
```

Returns the score bias against flipping a mover. The score bias is determined by the scoring system but can be controlled indirectly with the *SetFlipBiasScale* method.

See also

- *GetFlipBiasScale* method

GetFlipBiasScale

```
double GetFlipBiasScale() const
```

Returns the scaling factor for the score bias against flipping a mover. The actual score bias is determined by the scoring system.

See also

- *GetFlipBias* method

GetFlipPredicate

```
const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>* GetFlipPredicate() const
```

Returns a new copy of the predicate that *OEPlaceHydrogens* will use to restrain to the flipped state all flippable movers having a matching atom.

(Default: *OEIsFalseAtom*)

See also

- *SetFlipPredicate* method
- *GetBypassPredicate* method
- *GetNoFlipPredicate* method

GetMaxSubstateCutoff

```
double GetMaxSubstateCutoff() const
```

Max number of substates processed in a single step of the state optimizer.

See also

- *SetMaxSubstateCutoff* method

GetNoFlipPredicate

```
const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>* GetNoFlipPredicate() const
```

Returns a new copy of the predicate that *OEPlaceHydrogens* will use to restrain to the original (unflipped) state all flippable movers having a matching atom.

(Default: *OEIsFalseAtom*)

See also

- *SetNoFlipPredicate* method
- *GetBypassPredicate* method
- *GetFlipPredicate* method

GetStandardizeBondLen

```
bool GetStandardizeBondLen() const
```

Returns whether the length of bonds to hydrogen atoms are adjusted to standard size.

GetWaterProcessing

```
unsigned GetWaterProcessing() const
```

Returns a constant indicating how waters (and analogs) are processed.

See also

- *OEPlaceHydrogensWaterProcessing* namespace

SetAddMetadata

```
void SetAddMetadata()
```

Controls whether metadata describing the results should be added to the molecule. The metadata is the string generated by the *OEPlaceHydrogensDetails.Describe* method and can be accessed by the *OEGetPlaceHydrogensMetadata* function.

(Default: `true`)

See also

- *OEHasPlaceHydrogensMetadata* function
- *OEGetPlaceHydrogensMetadata* function
- *OEClearPlaceHydrogensMetadata* function
- *OEPlaceHydrogensDetails* class

SetAltsMustBeCompatible

```
void SetAltsMustBeCompatible()
```

Set if alternate location codes are required to be compatible (either identical or blank) for atoms to interact. If set to `true`, incompatible atoms will not interact.

The appropriate setting will depend on how alternate locations are prepared in the input molecule. If an *OEAltLocationFactory* was used to select the highest occupancy alternate locations, only one location remains for each atom so `compatible` should be `false`, it could prevent legitimate interactions. The same is true if alt information was stripped and only codes `A` and *blank* remain (the default behavior when a `.pdb` file is read with the default flavor).

If all location codes are retained on input with the *ALTLOC* flavor and not modified, then `compatible` should be `true`. This is not perfect, but generally works OK. But it is generally recommended that you read `.pdb` files using the *ALTLOC* flavor, process it with the *OEAltLocationFactory*, and leave `compatible` set to `false` when calling *OEPlaceHydrogens*.

(Default: `false`)

See also

- *Constructors*
- *OEIFlavor.PDB.ALTLOC* flavor
- *OEAltLocationFactory* class

SetBadClashOverlapDistance

```
bool SetBadClashOverlapDistance(double overlap)
```

Sets the amount of excess van der Waals overlap considered to be a "bad" clash. (Default: 0.4 Å)

SetBypassPredicate

```
bool SetBypassPredicate(const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred)
```

Stores a predicate that *OEPlaceHydrogens* will use to exclude from analysis all movers having a matching atom. Useful for groups where a specific orientation has already been established.

Note

For any given mover, this predicate takes precedence over both the *flip* and the *noFlip* predicates.

See also

- *GetBypassPredicate* method
- *SetFlipPredicate* method
- *SetNoFlipPredicate* method

SetFlipBiasScale

```
bool SetFlipBiasScale(double scale)
```

Sets a scaling factor for the score bias against flipping a mover. The actual score bias is determined by the scoring system. Setting the bias to `0.0` will turn off any bias towards the input structure. (Default: `1.0`)

See also

- *GetFlipBias* method

SetFlipPredicate

```
bool SetFlipPredicate(const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred)
```

Stores a predicate that *OEPlaceHydrogens* will use to restrain to the flipped state all flippable movers having a matching atom.

Note

For any given mover, if either the *bypass* predicate or the *noFlip* predicate matches, this predicate will be ignored.

See also

- *GetFlipPredicate* method
- *SetBypassPredicate* method
- *SetNoFlipPredicate* method

SetMaxSubstateCutoff

```
bool SetMaxSubstateCutoff(double maxStates)
```

Set the maximum number of substates processed in a single step of the state optimizer. (Default: 1.0e8)

See also

- *GetMaxSubstateCutoff* method

SetNoFlipPredicate

```
bool SetNoFlipPredicate(const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred)
```

Stores a predicate that *OEPlaceHydrogens* will use to restrain to the original (unflipped) state all flippable movers having a matching atom. Useful in preventing a group from being automatically flipped.

Note

For any given mover, if the *bypass* predicate matches, this predicate will be ignored. In addition, this predicate takes precedence over the *flip* predicate.

See also

- *GetNoFlipPredicate* method
- *SetBypassPredicate* method
- *SetFlipPredicate* method

SetStandardizeBondLen

```
void SetStandardizeBondLen()
```

Sets whether the length of bonds to hydrogen atoms are adjusted to standard size. (Default: `true`)

SetWaterProcessing

```
bool SetWaterProcessing(unsigned process)
```

Sets how waters (and analogs) are processed. (Default: *OEPlaceHydrogensWaterProcessing.Default*)

See also

- *OEPlaceHydrogensWaterProcessing* namespace

OEProteinInteractionHintComponent

```
class OEProteinInteractionHintComponent : public OEInteractionHintComponentTypeBase
```

Built-in component type that classifies molecules as being 'protein' when stored in a *OEInteractionHintContainer* object.

The following methods are publicly inherited from *OEInteractionHintComponentTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OEProteinInteractionHintComponent()
```

Default constructor.

```
OEProteinInteractionHintComponent(const OEProteinInteractionHintComponent& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintComponentTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEProteinInteractionHintComponent* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEChem::OEMolBase &) const
```

Returns whether a molecule can be inserted to a *OEInteractionHintContainer* object with a *OEProteinInteractionHintComponent* type *i.e.* it returns true if the molecule has 3D coordinates.

OERamachandranAnalysis

```
class OERamachandranAnalysis
```

This class represents an *OERamachandranAnalysis*, which holds information about the backbone conformation, of a protein residue, which can be used for plotting a Ramachandran plot. A Ramachandran plot, is a plot of the protein backbone ϕ and ψ angles of an amino acid residue, on the x- and y-axis, respectively. The diagram was originally developed in 1963 by G. N. Ramachandran [Ramachandran-1963] to visualizes energetically accessible regions of protein conformational space. Initially, there was a single plot used for all standard amino acid residues, but recent work has been done on certain residue types that do not follow the general pattern [Lovell-2003] [Chen-2010] [Hintze-2016], specifically new diagrams have been derived for: glycine, isoleucine and valine, cis-proline, trans-proline, and for residues preceding proline residues, since their backbone conformational space is affected by the following proline residue. The data was extracted from the open-source Computational Crystallography Toolbox (CCTBX) [Grosse-Kunstleve-2002].

Note

The function expects alternate conformations of the protein to be collapsed, meaning only a single alternate conformation should exist in the molecule object

The following is an example of determining the Ramachandran classification for each standard residue in a protein:

1. Read in a PDB and collapse the alternate conformations (if present). See chapter How to Correctly Read a Macromolecular File Format (mmCIF/PDB) in **Spruce TK**.
2. Loop over the CA atoms in the protein molecule. Alternatively, (not shown) loop over the residues in the protein.
3. For each CA atom or residue construct the *OERamachandranAnalysis*
4. Print out information about outliers for further analysis

Listing 1: Example of Ramachandran outlier analysis

```
        // Loop over the CA atoms in the protein
        foreach(OEAtomBase atom in mol.GetAtoms(new OEIsCAlpha()))
        {
            OERamachandranAnalysis rama = new OERamachandranAnalysis(atom);
            // Print out information about outliers for further analysis
            if (rama.GetRamaCategory() == OERamaCategory.Outlier)
            {
```

(continues on next page)

(continued from previous page)

```
                OEResidue res = OEChem.OEAtomGetResidue(atom);
                Console.WriteLine("Found: " + OEBio.OEGetRamachandranCategoryName(rama.
↪GetRamaCategory()));
                Console.WriteLine("  Residue: " + res.GetName() + " " + res.
↪GetResidueNumber() + " " + res.GetChainID());
                Console.WriteLine("  Type: " + OEBio.OEGetRamachandranTypeName(rama.
↪GetRamaType()) + ", Score: " + rama.GetRamaScore());
            }
        }
```

See also

- *OERamaCategory* namespace
- *OERamaType* namespace
- OERenderRamachandranPlot function in **Grapheme TK** to visualize Ramachandran plots

There is no image available for the PDF version of this book.

Constructors

```
OERamachandranAnalysis(const OEChem::OEAtomBase* atom)
```

The atom based constructor accepts any atom of a given amino acid residue and returns an *OERamachandranAnalysis*. While any atom from a residue is acceptable, it is advised to loop over the protein atoms with a predicate selecting e.g. only CA atoms, as this ensures the analysis is only done once per residue. The molecule must have 3D coordinates assigned. See code example above.

```
OERamachandranAnalysis(const OEHierResidue& res)
OERamachandranAnalysis(const OEChem::OEMolBase& mol,
                       const OEChem::OEResidue& res)
```

Residue based constructors. These constructors, like the atom based one, use *OEGetTorsion*, to determine the ϕ, ψ, and ω dihedrals angles. The dihedral is used to differentiate Cis/Trans proline. The molecule must have 3D coordinates assigned.

```
OERamachandranAnalysis(double phi, double psi,
                       const unsigned int ramaType)
```

Constructor that determines the Ramachandran category based on raw data, where the user specifies which underlying Ramachandran plot to use, by specifying the type using the *OERamaType* namespace.

```
OERamachandranAnalysis(const OERamachandranAnalysis& rhs)
```

Copy constructor.

GetPhi

```
double GetPhi() const
```

Returns the value of the backbone ϕ dihedral angle used to evaluate the score, in radians.

GetPsi

```
double GetPsi() const
```

Returns the value of the backbone ψ dihedral angle used to evaluate the score, in radians.

GetRamaCategory

```
unsigned int GetRamaCategory() const
```

Returns a value corresponding to the classification for the residue from the *OERamaCategory* namespace.

GetRamaScore

```
double GetRamaScore() const
```

Returns the value of the residue's Ramachandran score, which is a score relative to contours of expected Ramachandran behavior.

GetRamaType

```
unsigned int GetRamaType() const
```

Returns a value corresponding to the *OERamaCategory* namespace of the type of Ramachandran plot data used to determine the classification.

OEReceptor

```
class OEReceptor
```

The *OEReceptor* represents a receptor as used in a **OEDocking** or **POSIT** calculation.

The *OEReceptor* class defines the following public methods:

- *AddProteinConstraint*
- *Clear*
- *ClearCustomConstraints*
- *ClearProteinConstraint*
- *ClearProteinConstraints*
- *GetCustomConstraints*
- *GetInnerContourLevel*
- *GetNegativeImageGrid*
- *GetOuterContourLevel*
- *GetOuterContourVolume*
- *GetInnerContourVolume*
- *NumProteinConstraints(*
- *GetProteinConstraints*

- *GetTargetMask*
- *GetTargetPred*
- *IsValid*
- *HasCustomConstraints*
- *HasInnerContourLevel*
- *HasNegativeImageGrid*
- *HasOuterContourLevel*
- *HasProteinConstraints*
- *SetCustomConstraints*
- *SetInnerContourLevel*
- *SetOuterContourLevel*

Constructor

```
OEReceptor()
OEReceptor(const OEReceptor &)
```

Default and copy constructors.

Assignment operator.

AddProteinConstraint

```
bool AddProteinConstraint(const OEReceptorProteinConstraint&)
```

Adds a protein constraint. If the receptor already has a constraint on the specified atom the constraint will be replaced, otherwise a new one will be created.

Clear

```
bool Clear()
```

Clear all receptor information from receptor, and makes it an invalid receptor.

ClearCustomConstraints

```
bool ClearCustomConstraints()
```

Clear all custom constraints from the receptor.

ClearProteinConstraint

```
bool ClearProteinConstraint(const OEReceptorProteinConstraint*)
```

Remove the specified protein constraint from the receptor.

ClearProteinConstraints

```
bool ClearProteinConstraints()
```

Clear all protein constraint from the receptor.

GetCustomConstraints

```
const OEReceptorCustomConstraints& GetCustomConstraints() const
OEReceptorCustomConstraints& GetCustomConstraints()
```

Return a copy of the receptor's custom constraints.

GetInnerContourLevel

```
float GetInnerContourLevel() const
```

Return the inner contour level of the receptor

GetNegativeImageGrid

```
OESystem::OEGrid<float> GetNegativeImageGrid() const
```

Return a copy of the receptor's negative image potential grid.

GetOuterContourLevel

```
float GetOuterContourLevel() const
```

Return the outer contour level of the receptor

GetOuterContourVolume

```
float GetOuterContourVolume() const
```

Return the volume of the outer contour of the receptor at the set contour level. Will return zero if the contour is not set or has a negative value.

GetInnerContourVolume

```
float GetInnerContourVolume() const
```

Return the volume of the inner contour of the receptor at the set contour level Will return zero if the contour is not set or has a negative value.

NumProteinConstraints

```
unsigned int NumProteinConstraints(bool enabledOnly = true) const
```

Returns the number of *OEReceptorProteinConstraint* objects contained by this class.

If *enabledOnly* is true then only *OEReceptorProteinConstraint* objects for which the method *OEReceptorProteinConstraint.GetEnabled* returns true will be counted.

GetProteinConstraints

```
OESystem::OEIterBase<const OEReceptorProteinConstraint>*␣
↪GetProteinConstraints(bool enabledOnly = true) const
OESystem::OEIterBase<OEReceptorProteinConstraint>* GetProteinConstraints(bool␣
↪enabledOnly = true)
```

Returns an iterator over the receptor's protein constraints. If `enabledOnly` is `true` then only enabled constraints will be returned.

GetTargetMask

```
unsigned GetTargetMask() const
```

Return the target mask corresponding to the receptor. The mask and predicate, can be used with the *OEDesignUnit.GetComponents* method to generate the molecule used in receptor generation.

GetTargetPred

```
OEBio::OEAtomMatchResidue GetTargetPred() const
```

Returns the residue based *OEAtomMatchResidue* predicate used with the corresponding target mask to generate the receptor. The mask and predicate, can be used with the *OEDesignUnit.GetComponents* method to generate the molecule used in receptor generation.

See also

- *OEAtomMatchResidue* class
- *OEDesignUnit.GetComponents* method

IsValid

```
bool IsValid() const
```

Returns if the this is a valid receptor, and ready to be used for **OEDocking** or **POSIT** calculation.

HasCustomConstraints

```
bool HasCustomConstraints(const bool enabledOnly = true) const
```

Returns true if receptor has custom constraints. If `enabledOnly` is `true` then at least one of the constraints must be enabled.

HasInnerContourLevel

```
bool HasInnerContourLevel() const
```

Returns true if the receptor has an inner contour level set.

HasNegativeImageGrid

```
bool HasNegativeImageGrid() const
```

Returns true if the receptor has negative image potential grid. A receptor is invalid without the negative image potential grid.

HasOuterContourLevel

```
bool HasOuterContourLevel() const
```

Returns true if the receptor has an outer contour level set.

HasProteinConstraints

```
bool HasProteinConstraints(const bool enabledOnly = true) const
```

Returns true if receptor has protein constraints. If `enabledOnly` is `true` then at least one of the constraints must be enabled.

SetCustomConstraints

```
bool SetCustomConstraints(const OEReceptorCustomConstraints&)
```

Set **custom constraints** to the receptor. Returns `true` is set.

SetInnerContourLevel

```
bool SetInnerContourLevel(const float level)
```

Set **inner contour level** to the receptor. Returns `true` is set.

SetOuterContourLevel

```
bool SetOuterContourLevel(const float level)
```

Set **outer contour level** to the receptor. Returns `true` is set.

OEReceptorConstraintFeature

```
class OEReceptorConstraintFeature
```

OEBio_OEReceptorConstraintFeature represents a single custom docking constraint in an *OEReceptor*.

To satisfy a docking constraint a pose must either have:

1. At least one atom that matches one of the SMARTS patterns returned by *GetSmarts* **and** falls within one of the spheres returned by *GetFeatureSpheres*.
2. At least one heavy atom that falls within one of the spheres returned by *GetFeatureSpheres*, **if** the *OEReceptorConstraintFeature* has no SMARTS patterns.

Constructor

```
OEReceptorConstraintFeature()
OEReceptorConstraintFeature(const OEReceptorConstraintFeature&)
```

Default and copy constructors.

GetFeatureSpheres

```
OESystem::OEIterBase<OEMath::OESphere>* GetFeatureSpheres()
OESystem::OEIterBase<OEMath::OESphere>* GetFeatureSpheres()
```

Returns all spheres associated with this custom constraint feature.

AddNewSphere

```
OEMath::OESphere* AddNewSphere()
OEMath::OESphere* AddNewSphere(const OEMath::OESphere& sph)
```

Adds an *OESphere* object to this class. If *sph* is passed the created sphere will be a copy of *sph* otherwise a default constructed sphere will be created.

Note that the returned memory is owned by this class and will be destroyed by the destructor of this class.

DeleteSphere

```
bool DeleteSphere(const OEMath::OESphere* sph)
```

Deletes a *OESphere* held by this class. Note that after this call *sph* will no longer point to a valid *OESphere* object.

NumSpheres

```
unsigned int NumSpheres() const
```

Returns the number of spheres held by this object.

ClearSpheres

```
bool ClearSpheres()
```

Returns this object to its default constructed state. Deleting all spheres and SMARTS patterns it currently holds.

GetSmarts

```
OESystem::OEIterBase<const std::string>* GetSmarts() const
OESystem::OEIterBase<std::string>* GetSmarts()
```

Returns an iterator over all SMARTS patterns associated with this *OEReceptorConstraintFeature*.

AddSmarts

```
bool AddSmarts(const std::string& smarts)
```

Adds a SMARTS pattern to this *OEReceptorConstraintFeature*.

DeleteSmarts

```
bool DeleteSmarts(const std::string& smarts)
```

Deletes the specified SMARTS pattern from this *OEReceptorConstraintFeature*.

NumSmarts

```
unsigned int NumSmarts() const
```

Returns the number of SMARTS patterns held by this *OEReceptorConstraintFeature*.

ClearSmarts

```
bool ClearSmarts()
```

Deletes all SMARTS patterns associated with this *OEReceptorConstraintFeature*.

SetFeatureName

```
void SetFeatureName(const std::string&)
```

Sets the name of this feature.

GetFeatureName

```
std::string GetFeatureName() const
```

Returns the name of the feature.

SetEnabled

```
void SetEnabled(const bool enabled)
```

Sets whether this feature is enabled or not. If the feature is disabled the constraint will be ignored during the docking process.

GetEnabled

```
bool GetEnabled() const
```

Returns whether this feature is enabled. If this function returns false this constraint will be ignored during the docking process.

CreateCopy

```
OEReceptorConstraintFeature* CreateCopy() const
```

Creates a copy of this *OEReceptorConstraintFeature*. Note that the memory returned by this function is **not** owned by this class, and is the responsibility of the calling function.

OEReceptorCustomConstraints

```
class OEReceptorCustomConstraints
```

OEReceptorCustomConstraints holds a set of custom docking constraint features. Each custom constraint is defined by an *OEReceptorConstraintFeature* contained by this class.

Constructors

```
OEReceptorCustomConstraints()
OEReceptorCustomConstraints(const OEReceptorCustomConstraints& )
```

Default and copy constructors.

GetConstraintFeatures

```
OESystem::OEIterBase<OEReceptorConstraintFeature>* GetConstraintFeatures(bool␣
↪enabledOnly = true)
OESystem::OEIterBase<const OEReceptorConstraintFeature>* GetConstraintFeatures(bool␣
↪enabledOnly = true) const
```

Returns an iterator over all features contained by this class, or all enabled features if *enabledOnly* is true (see *OEReceptorConstraintFeature.GetEnabled* method).

AddNewFeature

```
OEReceptorConstraintFeature* AddNewFeature()
```

Adds a new *OEReceptorConstraintFeature* object to this class and returns a pointer to the newly created *OEReceptorConstraintFeature*. The returned *OEReceptorConstraintFeature* object is owned by this class, and will be destroyed by the destructor of this class.

DeleteFeature

```
bool DeleteFeature(const OEReceptorConstraintFeature* feature)
```

Deletes an *OEReceptorConstraintFeature* owned by this class. Note that after this method is called *feature* will no longer point to a valid *OEReceptorConstraintFeature* object.

NumFeatures

```
unsigned int NumFeatures(bool enabledOnly = true) const
```

Returns the number of *OEReceptorConstraintFeature* objects contained by this class.

If *enabledOnly* is true then only *OEReceptorConstraintFeature* objects for which the method *OEReceptorConstraintFeature.GetEnabled* returns true will be counted.

Clear

```
bool Clear()
```

Returns this object to its default constructed state, deleting any *OEReceptorConstraintFeature* objects it may currently be holding.

OEReceptorProteinConstraint

```
class OEReceptorProteinConstraint
```

This class holds information about a single receptor protein constraint.

GetAtom

```
OEChem::OEAtomBase* GetAtom(const OEChem::OEMolBase& mol) const
```

Return the protein atom of the receptor the constraint is associated with. The input molecule refers to the corresponding target protein.

GetType

```
unsigned int GetType() const
```

Type for the constraint (see *OEProteinConstraintType*).

GetEnabled

```
bool GetEnabled() const
```

Returns true if the constraint is enabled and false otherwise.

GetName

```
std::string GetName() const
```

Returns the name of the constraint.

SetAtom

```
void SetAtom(OEChem::OEAtomBase* atom)
```

Sets the atom of the receptor the constraint will be associated with.

SetType

```
void SetType(unsigned int type)
```

Sets the constraint type (see *OEProteinConstraintType*).

SetName

```
void SetEnabled(bool enabled)
```

Sets the name of the constraint.

SetEnabled

```
void SetName(std::string name)
```

Sets the enabled flag of the constraint.

OEResidueCategoryData

```
class OEResidueCategoryData
```

This class represents a container for a collection of residue name lists, each list associated with a category in the *OEResidueDatabaseCategory* namespace. These lists define various types of components found in a macromolecular complex. The main use of these lists is to identify molecules that are *not* ligands, so that ligands may be recognized more easily.

The *OEMolComplexCategorizer* class makes use of a *OEResidueCategoryData* object. The following *OEMolComplexCategorizer* methods get or set an *OEResidueCategoryData*:

GetResidueCategoryData	*SetResidueCategoryData*

Constructors

```
OEResidueCategoryData()
OEResidueCategoryData(const OEResidueCategoryData& rhs)
```

Constructor and copy constructor.

AddToDB

```
bool AddToDB(const unsigned category, const std::string& resName)
bool AddToDB(const unsigned category,
             const std::vector<std::string>& list)
```

Add an entry to the list associated with the specified category.

> **Note**
>
> The overload that takes a std::vector parameter is only available in C++.

ClearDBCategory

```
bool ClearDBCategory(const unsigned category)
```

Remove all entries in the list associated with the specified category.

RemoveFromDB

```
bool RemoveFromDB(const unsigned category,
                  const std::string& resName)
bool RemoveFromDB(const unsigned category,
                  const std::vector<std::string>& list)
```

Remove an entry from the list associated with the specified category.

Note

The overload that takes a std::vector parameter is only available in C++.

FoundInDB

```
bool FoundInDB(const unsigned category, const std::string& resName) const
```

Is the specified residue name found in the list associated with the specified category.

AllInDB

```
bool AllInDB(const unsigned category,
             const std::vector<std::string>& resNames) const
```

Are all the specified residue names found in the list associated with the specified category.

GetEntries

```
OESystem::OEIterBase<const std::string>* GetEntries(const unsigned category) const
```

Return an iterator of all the entries in the list associated with the specified category.

Size

```
unsigned Size(const unsigned category) const
```

Return the size of the list associated with the specified category.

OERotamer

```
struct OERotamer
```

This class represents an *OERotamer*, an abstraction of a frequently observed conformation of an amino-acid side-chain. Statistical surveys of side-chain conformations for well-curated sets of protein structures reveal that almost all can be assigned to a limited number of clusters. The mean, or more usefully the *mode*, of each cluster's dihedral angles can be used to represent the cluster as a whole. Collections of these dihedral angles, along with the observed probability of the associated cluster is known as a rotamer library.

See also

- *OESetRotamer* function that takes an *OERotamer* as an argument and modifies the 3-dimensional structure of a particular amino-acid side-chain so that the dihedral angles are that of the rotamer.
- *OEGetRotamers* function that returns an *OERotamer* iterator over each rotamer for a given amino-acid in a specific rotamer library.
- *OERotamerLibrary* namespace
- *OEGetTorsion* function

Note

If a specific side-chain dihedral angle does not exist in the given residue, the above GetChiN e.g. *OERotamer.GetChi1* methods return `-100.0`.

GetChi1

```
float GetChi1() const
```

Return the value of the side-chain χ_1 dihedral angle, `N-CA-CB-xG(1)`.

GetChi2

```
float GetChi2() const
```

Return the value of the side-chain χ_2 dihedral angle, `CA-CB-xG(1)-xD(1)`.

GetChi3

```
float GetChi3() const
```

Return the value of the side-chain χ_3 dihedral angle, `CB-xG-xD-xE(1)`.

GetChi4

```
float GetChi4() const
```

Return the value of the side-chain χ_4 dihedral angle, `xG-xD-xE-xZ`.

GetProbability

```
float GetProbability() const
```

Return the probability assigned to this rotamer by the rotamer library used to set the rotamer parameters (see *OERotamerLibrary*).

GetResidueIndex

```
unsigned int GetResidueIndex() const
```

Return the *OEResidueIndex* value indicating the type of amino-acid residue used to set the rotamer parameters.

OESaltBridgeInteractionHint

```
class OESaltBridgeInteractionHint : public OEInteractionHintTypeBase
```

The *OESaltBridgeInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as **salt-bridge** interactions (inter- and intramolecular).

See also

- *OESaltBridgeInteractionHintType* namespace for specific **salt-bridge** interaction types
- *OEIsSaltBridgeInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **salt-bridge** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its methods:
  - *OEPerceiveInteractionOptions.SetMaxSaltBridgeDistance*
  - *OEPerceiveInteractionOptions.GetExtendedSaltBridgeDistance*

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OESaltBridgeInteractionHint(unsigned int type)
```

type
This value has to be from the *OESaltBridgeInteractionHintType* namespace.

Note

Two *OESaltBridgeInteractionHint* objects with different types from the *OESaltBridgeInteractionHintType* namespace are considered to be different.

```
OESaltBridgeInteractionHint(const OESaltBridgeInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OESaltBridgeInteractionHint* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **salt-bridge** interaction when added to an *OEInteractionHintContainer* object.

OESequenceAlignment

```
class OESequenceAlignment
```

This class contains information regarding a sequence alignment between two molecules that have residue information. The class includes information concerning the methods used to generate the alignment as well as *OEMatchBase* information for either the alpha carbons or all the backbone atoms that are paired by the alignment. The *OESequenceAlignment* class can be passed to an an *OERMSD* function to generate 3D alignment based on the sequence alignment, and can also be passed to the *OEWriteAlignment* function to generate a standard sequence alignment text output.

Constructors

```
OESequenceAlignment()
OESequenceAlignment(const OESequenceAlignment& rhs)
```

Default and copy constructors.

operator bool

```
operator bool() const
```

An *OESequenceAlignment* object is considered invalid until its implementation is set.

Clear

```
void Clear()
```

Resets the object to the state in which it is default constructed.

GetBackboneMatch

```
const OEChem::OEMatchBase& GetBackboneMatch(unsigned idx1 = 0,
                                            unsigned idx2 = 1)
```

Returns an *OEMatchBase* that is the alignment of backbone atoms between the sequences indicated by 'idx1' and 'idx2'. By default, the indices are *0* and *1* respectively, corresponding to a simple pairwise alignment.

GetCAlphaMatch

```
const OEChem::OEMatchBase& GetCAlphaMatch(unsigned idx1 = 0,
                                          unsigned idx2 = 1)
```

Returns an *OEMatchBase* that is the alignment of alpha carbon atoms between the sequences indicated by 'idx1' and 'idx2'. By default, the indices are *0* and *1* respectively, corresponding to a simple pairwise alignment.

GetChainID

```
char GetChainID(unsigned int idx) const
```

Returns the chain id of the given sequence.

idx
The index of the sequence. In a simple pairwise alignment this should be either *0* or *1*.

GetExtend

```
int GetExtend() const
```

Returns an integer that is the extension penalty used in creating the sequence alignment.

GetGap

```
int GetGap() const
```

Returns an integer that is the gap penalty used in creating the sequence alignment.

GetMaxSeqIdx

```
unsigned GetMaxSeqIdx() const
```

Returns a pointer that is the maximum index of any of the sequences used in the alignment.

GetMethod

```
unsigned GetMethod() const
```

Returns the unsigned integer that corresponds to one of the constants in the namespace *OESeqAlignmentMethod*. This is the method that was used to create the alignment.

GetScore

```
int GetScore() const
```

Returns an integer that is the score of the sequence alignment.

GetSequenceIdentity

```
double GetSequenceIdentity() const
```

Returns the percentage identity between the two sequences.

GetSequenceSimilarity

```
double GetSequenceSimilarity() const
```

Returns the percentage similarity between the two sequences based on the given method.

See also

- *OESeqAlignmentMethod* namespace

OESplitMolComplexOptions

```
class OESplitMolComplexOptions
```

The *OESplitMolComplexOptions* class encapsulates properties and resources that determine how *OESplitMolComplex* and *OEGetMolComplexComponents* operate. For examples detailing how to modify and/or operate different filters see *Splitting Macromolecular Complexes*.

Constructors

```
OESplitMolComplexOptions(const OESplitMolComplexOptions& rhs)
OESplitMolComplexOptions(const std::string& ligName = std::string())
```

Constructor and copy constructor. If `ligName` is specified in the call to the constructor, components will be recognized as ligands if and only if they have a matching residue name string.

See also

- *OEMolComplexCategorizer.Constructors*

GetAddRGroup

```
bool GetAddRGroup() const
```

Returns `true` if "R groups" are to be added when covalent bonds are broken.

GetAdjustHCount

```
bool GetAdjustHCount() const
```

Returns `true` if the implicit hydrogen count is to be adjusted when covalent bonds are broken.

GetAssumptions

```
unsigned GetAssumptions() const
```

Return assumptions about molecular perception.

See also

- *OEAssumption*

GetCategorizer

```
OEMolComplexCategorizerBase* GetCategorizer() const
```

Return a new copy of the *OEMolComplexCategorizerBase* object used to categorize molecular components.

GetCollectProteinByChain

```
bool GetCollectProteinByChain() const
```

Returns `true` if fragmented protein chains be associated with the same the same site.

GetCombinedFilter

```
OESystem::OEUnaryPredicate<OESystem::OERoleSet>* GetCombinedFilter() const
```

Returns a new copy of the filter that matches the *ligand* or *protein* or *water* filters.

See also

- *OESplitMolComplexOptions.GetLigandFilter*
- *OESplitMolComplexOptions.GetProteinFilter*
- *OESplitMolComplexOptions.GetWaterFilter*
- *OESplitMolComplexOptions.GetOtherFilter*

GetLigandFilter

```
OESystem::OEUnaryPredicate<OESystem::OERoleSet>* GetLigandFilter() const
```

Returns a new copy of the *ligand* filter.

GetMaxBindingSiteDist

```
double GetMaxBindingSiteDist() const
```

Return the maximum distance from the ligand to be associated with a binding site. Can not be less than 1.0 Å.

GetMaxSurfaceWaterDist

```
double GetMaxSurfaceWaterDist() const
```

Return the maximum distance from the protein to be considered a surface water. Can not be less than 1.0 Å.

GetOtherFilter

```
OESystem::OEUnaryPredicate<OESystem::OERoleSet>* GetOtherFilter() const
```

Returns a new copy of the filter that matches everything that does *not* match the combined filter.

See also

- *OESplitMolComplexOptions.GetCombinedFilter*

GetPromoteUnknowns

```
bool GetPromoteUnknowns() const
```

Returns `true` if components classified as *OEMolComplexFilterCategory.Unknown* should be considered the ligand if no others ligand is found.

GetProteinFilter

```
OESystem::OEUnaryPredicate<OESystem::OERoleSet>* GetProteinFilter() const
```

Returns a new copy of the *protein* filter.

GetSeparateResidues

```
bool GetSeparateResidues() const
```

Returns `true` if proteins are to be separated into individual residues before selection. Separating into residues can be used to extract binding site residues.

GetSplitCovalent

```
bool GetSplitCovalent() const
```

Returns `true` if covalent ligands are to be searched for and split out as separate components.

GetSplitCovalentCofactors

```
bool GetSplitCovalentCofactors() const
```

Returns `true` if covalent cofactors are to be searched for and split out as separate components.

GetWarnNoLigand

```
bool GetWarnNoLigand() const
```

Returns `true` if a verbose level warning will be generated when a ligand is not found.

GetWaterFilter

```
OESystem::OEUnaryPredicate<OESystem::OERoleSet>* GetWaterFilter() const
```

Returns a new copy of the *water* filter.

ResetFilters

```
void ResetFilters(unsigned site = 1u, bool surfaceWaters = false,
                  unsigned ligCategory = OEMolComplexFilterCategory::Ligand,
                  unsigned protCategory = OEMolComplexFilterCategory::ProtComplex,
                  unsigned waterCategory = OEMolComplexFilterCategory::Water,
                  unsigned modelNo = 1u)
```

Regenerate filters based on parameters. Can be overridden by methods to set individual filters. Site 0 means all sites. ModelNo 0 means all models.

Note

It may be necessary to remove the input flavor *OEIFlavor.PDB.ENDM* to read all models (see the *splitmolcomplex* example for how to do this). In most situations, the `modelNo` parameter should be set to either 0 or 1 (the default). The reason for this is that the site counter, which the `site` parameter selects based on, does not reset for each model and therefore site and model are not independent. A good example of where this can be used is with a PDB biological assembly file: setting the `modelNo` parameter to 0 (use all models), allows a binding site spanning symmetry related proteins marked as different models to be recognized.

See also

- *OEMolComplexFilterCategory* namespace
- *OEMolComplexFilterFactory* function
- *OESplitMolComplexOptions.SetLigandFilter*
- *OESplitMolComplexOptions.SetProteinFilter*
- *OESplitMolComplexOptions.SetWaterFilter*
- *OEMolComplexFilter* class
- *Splitting Macromolecular Complexes* section

SetAddRGroup

```
void SetAddRGroup(bool addR = true)
```

Set whether “R groups” are to be added when covalent bonds are broken.

SetAdjustHCount

```
void SetAdjustHCount(bool adjust = true)
```

Set whether the implicit hydrogen count is to be adjusted when covalent bonds are broken.

SetAssumptions

```
void SetAssumptions(unsigned assume)
```

Set assumptions about molecular perception. Default = *OEAssumption.ResPerceived*

SetCategorizer

```
void SetCategorizer(const OEMolComplexCategorizerBase& cat)
```

Set the *OEMolComplexCategorizerBase* object used to categorize molecular components.

SetCollectProteinByChain

```
void SetCollectProteinByChain(bool collect = true)
```

Set whether fragmented protein chains should be collected into the same site. Default = `true`

SetLigandFilter

```
void SetLigandFilter(const OESystem::OEUnaryPredicate<OESystem::OERoleSet>& f)
```

Set the *ligand* filter. Default is an *OEMolComplexFilterFactory* created with *OEMolComplexFilterCategory.Ligand*

See also

- *OEMolComplexFilter* class
- *OEMolComplexFilterFactory* function
- *OESplitMolComplexOptions.ResetFilters*

SetMaxBindingSiteDist

```
void SetMaxBindingSiteDist(double dist)
```

Set the maximum distance from the ligand to be associated with a binding site. Default = 8.0 Å

SetMaxSurfaceWaterDist

```
void SetMaxSurfaceWaterDist(double dist)
```

Set the maximum distance from the protein to be considered a surface water. Default = 5.0 Å

SetPromoteUnknowns

```
void SetPromoteUnknowns(bool promote = true)
```

Set whether components classified as *OEMolComplexFilterCategory.Unknown* should be considered the ligand if no others ligand is found. Default = `true`

SetProteinFilter

```
void SetProteinFilter(const OESystem::OEUnaryPredicate<OESystem::OERoleSet>& f)
```

Set the *protein* filter. Default is an *OEMolComplexFilterFactory* created with *OEMolComplexFilterCategory.ProtComplex*

See also

- *OEMolComplexFilter* class
- *OEMolComplexFilterFactory* function
- *OESplitMolComplexOptions.ResetFilters*

SetSeparateResidues

```
void SetSeparateResidues(bool separate = true)
```

Set whether to separate proteins into individual residues before selection. Can be used to identify binding site residues. Default = `false`

SetSplitCovalent

```
void SetSplitCovalent(bool split = true)
```

Set whether covalent ligands are to be searched for and split out as separate components. Default = `false`

SetSplitCovalentCofactors

```
void SetSplitCovalentCofactors(bool split = true)
```

Set whether covalent cofactors are to be searched for and split out as separate components. Default = `false`

SetWarnNoLigand

```
void SetWarnNoLigand(bool warn = true)
```

Set whether to send out a verbose level warning if a ligand is not found. Default = `true`

SetWaterFilter

```
void SetWaterFilter(const OESystem::OEUnaryPredicate<OESystem::OERoleSet>& f)
```

Set the *water* filter. Default is an *OEMolComplexFilterFactory* created with *OEMolComplexFilterCategory.Water*

See also

- *OEMolComplexFilter* class
- *OEMolComplexFilterFactory* function
- *OESplitMolComplexOptions.ResetFilters*

OEStackingInteractionHint

```
class OEStackingInteractionHint : public OEInteractionHintTypeBase
```

The *OEStackingInteractionHint* class represents a built-in type that identifies interactions stored in an *OEInteractionHintContainer* object as Pi or T **stacking** interactions (inter- and intramolecular).

See also

- *OEStackingInteractionHintType* namespace for specific **stacking** interaction types
- *OEIsStackingInteractionHint* predicate
- *OEInteractionHint* class
- *OEInteractionHintContainer* class

To perceive **stacking** interactions:

- *OEPerceiveInteractionHints* function
- *OEPerceiveInteractionOptions* class and its methods:
  - *OEPerceiveInteractionOptions.SetMaxPiStackAngle*
  - *OEPerceiveInteractionOptions.SetMaxPiStackDistance*
  - *OEPerceiveInteractionOptions.SetMaxTStackAngle*
  - *OEPerceiveInteractionOptions.SetMaxTStackDistance*

The following methods are publicly inherited from *OEInteractionHintTypeBase*:

operator=	*operator==*	*GetName*
operator!=	*CreateCopy*	*IsValid*

Constructors

```
OEStackingInteractionHint(unsigned int type)
```

Creates an *OEStackingInteractionHint* object with the given type.

type
: This value has to be from the *OEStackingInteractionHintType* namespace.

Note

Two *OEStackingInteractionHint* objects with different types from the *OEStackingInteractionHintType* namespace are considered to be different.

```
OEStackingInteractionHint(const OEStackingInteractionHint& rhs)
```

Copy constructor.

CreateCopy

```
OEInteractionHintTypeBase* CreateCopy() const
```

Deep copy constructor that returns a copy of the object. The memory for the returned *OEStackingInteractionHint* object is dynamically allocated and owned by the caller.

IsValid

```
bool IsValid(const OEInteractionHintFragment *,
             const OEInteractionHintFragment *) const
```

Evaluates whether the two given fragments would form a valid **stacking** interaction when added to an *OEInteractionHintContainer* object.

It checks that each fragment contains at least five atoms, and the fragments belong to different molecules *i.e.* it is an **intermolecular** interaction.

OEStructureQuality

Attention

This API is currently available in C++ and Python.

```
class OEStructureQuality
```

This is a container class that holds all data relating to the structural quality of biomolecular system.

See also

- *OEIridiumData* class

Constructors

```
OEStructureQuality()
OEStructureQuality(const OEIridiumData& id)
OEStructureQuality(const OEStructureQuality& rhs)
```

Default and copy constructors.

operator bool

```
operator bool() const
```

Returns whether the *OEStructureQuality* object is valid.

Clear

```
void Clear()
```

Resets the OEStructureQuality object to its initial state.

GetIridiumData

```
OEIridiumData GetIridiumData()
const OEIridiumData GetIridiumData() const
```

Returns a copy of the *OEIridiumData* object stored within the *OEStructureQuality* object.

HasIridiumData

```
bool HasIridiumData() const
```

Checks that the *OEIridiumData* object has been set.

SetIridiumData

```
bool SetIridiumData(const OEIridiumData& id)
```

Sets the *OEIridiumData* object on the *OEStructureQuality* object. Returns `true` if successful, and `false` otherwise.

4.7.2 OEBio Constants

OEAltLocFactOption

This namespace contains the following constants used as options to *OEAltLocationFactory.Constructors*:

None

This constant indicates no options.

TrustInputBonds

This constant indicates that any existing bonds in the input molecule can be used. Missing bonds will still be perceived. Without this option, if any alternate location codes are found in the input mol, any bonds found will be ignored.

Since the basic bond perception mechanism will usually fail on a molecule that contains alternative conformation atoms, this option should only be used when special care has been taken with the input molecule (for example, if the molecule comes from *OEAltLocationFactory.GetSourceMol*).

OEAssumption

This namespace contains constants defining properties of molecules that can be assumed.

None

No assumptions should be made.

PDBOrder

Assume the molecules atoms are in standard PDB order.

BondedResidue

Assume that residues in the molecule have all the residues atoms bonded to one another.

ResPerceived

Assume that *OEResidues* have been perceived.

Default

Assume *PDBOrder* and *BondedResidue* and *ResPerceived*.

OECationPiInteractionHintType

This namespace contains constants for the *OECationPiInteractionHintType* interaction type.

See also

- *OEInteractionHint* class
- *OECationPiInteractionHint* class
- *OEIsCationPiInteractionHint* functor
- *OEIsIntermolecularCationPiInteractionHint* functor
- *OEIsIntramolecularCationPiInteractionHint* functor

LigandCation

Same as the *OECationPiInteractionHintType.ProteinPi* constant.

LigandPi

This type of Cation Pi interaction has a ligand pi ring system interacting with a Lysine, N-methyl Lysine or Arginine of the protein.

ProteinCation

Same as the *OECationPiInteractionHintType.LigandPi* constant.

ProteinPi

This type of Cation Pi interaction includes a protein pi ring system interacting with a Lysine-like or Arginine-like ligand cation.

IntramolecularLigand

A cation-pi interaction with the *OECationPiInteractionHintType.IntramolecularLigand* type represents an intramolecular cation-pi interaction between a cation and an aromatic ring within the **ligand**.

IntramolecularProtein

A cation-pi interaction with the *OECationPiInteractionHintType.IntramolecularProtein* type represents an intramolecular cation-pi interaction between a cation and an aromatic ring within the **protein**.

OEChelatorInteractionHintType

This namespace contains constants for the *OEChelatorInteractionHint* interaction type.

See also

- *OEInteractionHint* class
- *OEChelatorInteractionHint* class
- *OEIsChelatorInteractionHint* functor

LigandChelates

Chelation bonds with this type represent an interaction in which a ligand chelates a metal that is part of the protein.

ProteinChelates

Chelation bonds with this type represent an interaction in which a protein chelates a metal that is part of the ligand.

IntramolecularProtein

Chelation bonds with this type represent an interaction in which a protein chelates a metal (near the active site) that is part of the protein.

OEDesignUnitComponents

Attention

This API is currently available in C++ and Python.

This namespace contains constants representing the various molecular types that can be stored within an *OEDesignUnit*, see *Storage Components*.

The namespace also lists some combined constants for convenient use in various general applications, for example *TargetComplex*, see *Combined Components*

Note

When using these in command-line applications, the two word names in storage components below need to have an underscore (_) between the words, e.g. *OtherNucleics* should be *other_nucleics*. When relevant these are noted below.

Combined constants

All

All components.

TargetComplex

This is bit mask that combines the following, and is often used in modeling application *OEDesignUnitComponents.Protein*, *OEDesignUnitComponents.Nucleic*, *OEDesignUnitComponents.Cofactors*, *OEDesignUnitComponents.Solvent*, *OEDesignUnitComponents.Metals*, *OEDesignUnitComponents.CounterIons*, *OEDesignUnitComponents.Lipids*, *OEDesignUnitComponents.OtherProteins*, *OEDesignUnitComponents.OtherNucleics*, *OEDesignUnitComponents.OtherLigands*, and *OEDesignUnitComponents.OtherCofactors*

TargetComplexNoSolvent

This is bit mask that combines the following, and is often used in modeling application *OEDesignUnitComponents.Protein*, *OEDesignUnitComponents.Nucleic*, *OEDesignUnitComponents.Cofactors*, *OEDesignUnitComponents.Metals*, *OEDesignUnitComponents.Lipids*, *OEDesignUnitComponents.OtherProteins*, *OEDesignUnitComponents.OtherNucleics*, *OEDesignUnitComponents.OtherLigands*, and *OEDesignUnitComponents.OtherCofactors*

Default

This is bit mask that combines *OEDesignUnitComponents.Protein*, *OEDesignUnitComponents.Nucleic*, *OEDesignUnitComponents.Ligand*, *OEDesignUnitComponents.Cofactors*, *OEDesignUnitComponents.Solvent*, *OEDesignUnitComponents.Metals*, *OEDesignUnitComponents.CounterIons*, and *OEDesignUnitComponents.Lipids*.

ListComponents

This is a bit mask that combines *OEDesignUnitComponents.Cofactors*, *OEDesignUnitComponents.Excipients*, *OEDesignUnitComponents.Sugars*, *OEDesignUnitComponents.Polymers*, *OEDesignUnitComponents.PostTranslational*, *OEDesignUnitComponents.OtherProteins*, *OEDesignUnitComponents.OtherNucleics*, *OEDesignUnitComponents.OtherLigands*, and *OEDesignUnitComponents.OtherCofactors*.

MacroMolComponents

This is bit mask that combines *OEDesignUnitComponents.Protein*, *OEDesignUnitComponents.Nucleic*, *OEDesignUnitComponents.OtherProteins*, and *OEDesignUnitComponents.OtherNucleics*

MolComponents

This is bit mask that combines *OEDesignUnitComponents.Protein*, *OEDesignUnitComponents.Nucleic*, *OEDesignUnitComponents.Ligand*, *OEDesignUnitComponents.Solvent*, *OEDesignUnitComponents.Metals*, *OEDesignUnitComponents.CounterIons*, *OEDesignUnitComponents.Lipids*, *OEDesignUnitComponents.PackingResidues*, and *OEDesignUnitComponents.Undefined*

Storage constants

Cofactors

Cofactors (*i.e.* any heterogen molecule) within a short distance to the active site.

CounterIons

All counter ions.

In command line use, please use *counter_ions*

Excipients

All excipient molecules.

Ligand

The ligand.

Lipids

All lipid molecules.

Metals

All system metals that are not cofactors.

Nucleic

All nucleic acid molecules.

OtherCofactors

Any non-primary cofactors.

In command line use, please use *other_cofactors*

OtherLigands

Any non-primary ligands (*i.e.* ligands not in the active site).

In command line use, please use *other_ligands*

OtherNucleics

Any non-primary nucleic acids.

In command line use, please use *other_nucleics*

OtherProteins

Any non-primary proteins.

In command line use, please use *other_proteins*

PackingResidues

All packing residue molecules.

In command line use, please use *packing_residues*

Polymers

All polymers, excluding typical biological polymers like DNA and proteins.

PostTranslational

All components resulting from post-translation modifications (PTMs) that are not sugars or lipids.

In command line use, please use *post_translational*

Protein

The primary protein molecule.

Solvent

All solvent molecules.

Sugars

All sugar molecules.

Undefined

Molecules that cannot be categorized by *OEGetDesignUnitComponentID* are Undefined, an example would be residues specifically marked with residue name **UNK**.

Unknown

Molecules cannot be stored using this constant. The constant exists to indicate invalid use of IDs or string versions of the namespace.

OEDesignUnitFileType

Attention

This API is currently available in C++ and Python.

This namespace contains constants representing the various file types that can be read with *OEReadDesignUnit*.

OEDesignUnit

An OEDesignUnit file (.oedu) will be read as an *OEDesignUnit*.

Undefined

The file type is undefined.

OEHalogenBondInteractionHintType

This namespace contains constants for the *OEHalogenBondInteractionHint* interaction type.

See also

- *OEInteractionHint* class
- *OEHalogenBondInteractionHint* class
- *OEIsHalogenBondInteractionHint* functor
- *OEIsIntermolecularHalogenBondInteractionHint* functor
- *OEIsIntramolecularHalogenBondInteractionHint* functor

LigandElectrophile

A halogen bond with the *OEHalogenBondInteractionHintType.LigandElectrophile* type represents an interaction between a Chlorine, Bromine or Iodine on the protein and an electrophile on the ligand.

LigandNucleophile

A halogen bond with the *OEHalogenBondInteractionHintType.LigandNucleophile* type represents an interaction between a Chlorine, Bromine or Iodine on the protein and a nucleophile on the ligand.

ProteinElectrophile

A halogen bond with the *OEHalogenBondInteractionHintType.ProteinElectrophile* type represents an interaction between a Chlorine, Bromine or Iodine on the ligand and an electrophile on the protein.

ProteinNucleophile

A halogen bond with the *OEHalogenBondInteractionHintType.ProteinNucleophile* type represents an interaction between a Chlorine, Bromine or Iodine on the ligand and a nucleophile on the protein.

IntramolecularLigand

A halogen bond with the *OEHalogenBondInteractionHintType.IntramolecularLigand* type represents an intramolecular halogen bond between a halogen and an acceptor within the **ligand**.

IntramolecularProtein

A halogen bond with the *OEHalogenBondInteractionHintType.IntramolecularProtein* type represents an intramolecular halogen bond between a halogen and an acceptor within the **protein**.

OEHBondInteractionHintType

This namespace contains constants for the *OEHBondInteractionHint* interaction type.

See also

- *OEInteractionHint* class
- *OEHBondInteractionHint* class
- *OEIsHBondInteractionHint* functor

ClashAcceptorAcceptor

A hydrogen bond with the *OEHBondInteractionHintType.ClashAcceptorAcceptor* type represents a clashing interaction between a ligand acceptor atom and a protein acceptor atom.

ClashDonorDonor

A hydrogen bond with the *OEHBondInteractionHintType.ClashDonorDonor* type represents a clashing interaction between a ligand donor atom and a protein donor atom.

Clash

Warning

This is a deprecated API. Please use *OEHBondInteractionHintType.ClashAcceptorAcceptor* or *OEHBondInteractionHintType.ClashDonorDonor* instead.

IntramolecularLigand

A hydrogen bond with the *OEHBondInteractionHintType.LigandAccepts* type represents an interaction between a donor and acceptor on the ligand component.

IntramolecularProtein

A hydrogen bond with the *OEHBondInteractionHintType.ProteinAccepts* type represents an interaction between a donor and acceptor on the protein component.

LigandAccepts

A hydrogen bond interaction with the *OEHBondInteractionHintType.LigandAccepts* type represents an interaction where the ligand component of an active site accepts a proton donated by the protein component of the active site.

LigandDonates

A hydrogen bond interaction with the *OEHBondInteractionHintType.LigandDonates* type represents an interaction where the ligand component of an active site donates a proton accepted by the protein component of the active site.

NonIdealLigandAccepts

A hydrogen bond interaction with the *OEHBondInteractionHintType.NonIdealLigandAccepts* type represents an interaction where the bond between the ligand component and a proton donated by the protein component is not ideal.

NonIdealLigandDonates

A hydrogen bond interaction with the *OEHBondInteractionHintType.NonIdealLigandDonates* type represents an interaction where the bond between the ligand component and a proton accepted by the protein component is not ideal.

NonIdealProteinAccepts

Same as the *OEHBondInteractionHintType.NonIdealLigandDonates* constant.

NonIdealProteinDonates

Same as the *OEHBondInteractionHintType.NonIdealLigandAccepts* constant.

ProteinAccepts

Same as the *OEHBondInteractionHintType.LigandDonates* constant.

ProteinDonates

Same as the *OEHBondInteractionHintType.LigandAccepts* constant.

UnpairedLigandDonor

A interaction with the *OEHBondInteractionHintType.UnpairedLigandDonor* type represents a donor on the ligand component that has no paired acceptor.

UnpairedLigandAcceptor

A interaction with the *OEHBondInteractionHintType.UnpairedLigandAcceptor* type represents an acceptor on the ligand component that has no paired donor.

UnpairedProteinDonor

A interaction with the *OEHBondInteractionHintType.UnpairedProteinDonor* type represents a donor on the protein component that has no paired acceptor.

UnpairedProteinAcceptor

A interaction with the *OEHBondInteractionHintType.UnpairedProteinAcceptor* type represents an acceptor on the protein component that has no paired donor.

OEIridiumCategory

Attention

This API is currently available in C++ and Python.

This namespace contains constants that represent the Iridium classification. These classifications were taken from the 2012 paper by Warren *et al.*, which describes the Iridium scoring methodology, and includes a database of structures to be used in validation of structure-based drug discovery tools [Warren-2012].

HT

The structure is highly trustworthy.

MT

The structure is mildly trustworthy.

NA

Not applicable – no Iridium category has been assigned.

NT

The structure is not trustworthy.

See also

- *OEGetIridiumCategoryName* function

- *OEGetIridiumCategoryFromName* function
- Categorization section in **OESpruce TK**

OEMakeAltMolOption

This namespace contains the following constants used as options to one of the *OEAltLocationFactory.MakePrimaryAltMol*, *OEAltLocationFactory.MakeCurrentAltMol*, or *OEAltLocationFactory.MakeAltMol* methods:

None

This constant indicates no options.

RemoveAltCodes

This constant indicates that all alternate location codes should be set to blank in the new subset molecule.

OEMolComplexData

This namespace contains the following SD tags attached to components returned by *OEGetMolComplexComponents* specifying various properties. Any other roles a component is associated with are also be assigned SD tags (with a value of `true`). These SD tags can be removed with the function *OEClearMolComplexSDData*.

CovalentSDTag

Identifies a covalent ligand or cofactor.

CovalentLigandSDTag

Alias for *CovalentSDTag* for backward compatibility.

PartOrderSDTag

The sequence number of the molecular component in the order it was identified in the source molecule.

SDTagPrefix

Prefix for all SD tags used when splitting molecular complexes.

SiteNumberSDTag

The component is associated with at least one binding site. The value is the sequence number of the binding site(s).

SurfaceWatersNumberSDTag

The component is a surface water. The value is the sequence number of the associated binding site(s).

See also

- *OEGetMolComplexComponents* function
- *OEClearMolComplexSDData* function

OEMolComplexFilterCategory

This namespace contains constants used as parameters to *OEMolComplexRoleFactory*, *OEGetMolComplexFilterCategoryName*, *OEMolComplexFilterFactory*, and *OESplitMolComplexOptions.ResetFilters*. These constants refer to functional roles that may be associated with a component by *OESplitMolComplex* or *OEGetMolComplexComponents* and are used to construct filters (see *OEMolComplexFilter*) that select a desired subset of components. The class *OESplitMolComplexOptions* manages several of these filters; others can be generated as required.

Anything

Match anything.

ChainA

Match any component in chain 'A'.

Filters for other chains may be build by combining *OEMolComplexChainRoleFactory* with *OERoleMolComplexFilterFactory*.

Cofactor

Match any component identified as a cofactor.

CofactorAndLigand

Match any component in a special category for items that are either a cofactor or a ligand if no ligand is otherwise found.

CounterIon

Match any counter-ion (either a first-row metal or an halogen).

Covalent

Match any component identified as a covalent ligand.

DNA_RNA

Match any DNA or RNA molecule.

Ligand

Match any ligand.

Lipid

Match any lipids.

Metal

Match any metal.

Misc

Component has been identified as miscellaneous. These are otherwise unclassified compounds that are not to be considered ligands (for example buffers and excipients).

Model1

Match any component in model number 1 (the default if model information is not explicitly specified).

Filters for other chains may be build by combining *OEMolComplexModelRoleFactory* with *OERoleMolComplexFilterFactory*.

Nothing

Match nothing.

Nucleotide

Match any nucleotide.

Peptide

Match any poly-peptide.

ProtComplex

Match any component identified as being part of the protein complex (protein + cofactor).

Protein

Match any protein chain.

Saccharide

Match any component identified as a sugar.

Site1

Match any component identified as being part of binding site 1.

Filters for other chains may be build by combining *OEMolComplexSiteRoleFactory* with *OERoleMolComplexFilterFactory*.

Solution

Match any component identified as being part of the solution (water + counter-ion).

SurfaceWaters1

Match any surface waters identified as being part of binding site 1.

Filters for other chains may be build by combining *OEMolComplexSurfaceWatersRoleFactory* with *OERoleMolComplexFilterFactory*.

Underdefined

Match any component containing a dummy atom (*OEElemNo.Du*).

Unknown

Match any component that would have been identified as a ligand but has a residue name that differs from that explicitly specified.

Water

Match any water molecule.

OEPlaceHydrogensMoverClass

This namespace contains constants used by the *OEPlaceHydrogensMoverNote* class to describe each "mover" (movable functional group processed by `OEPlaceHydrogens`).

AmideNO

Terminal amide group, as in asparagine or glutamine (flippable).

AroN

Aromatic nitrogen other than an imidazole, deprotonated when coordinating a metal.

AroCH3

Aromatic methyl group. Has two rotational states, always placing one methyl hydrogen in the ring plane.

Background

Collection of non-moving atoms surrounding another mover. For waters, other movers are treated as background.

COOH

Protonated carboxylic acid (or analog). Hydrogen is rotated and group flipped.

DH

Donor with rotatable hydrogen, as in serene, theonine, tyrosine, cysteine, or ribose. Rotational states based on interactions with nearby hydrogen bond donors and acceptors.

Imidazole

Imidazole group, as in histidine (maximum of six states: two flip states times three protonation states HD1, HE2, and plus (+) charged with both HD1 & HE2). If between two metals, can instead be doubly deprotonated with a negative (-) charge.

Max

The total number of different mover classes.

N

Non-aromatic nitrogen deprotonated when coordinating a metal.

NHsp2Only

Secondary amine with a sp2 geometry.

NHsp3Only

Secondary amine with a sp3 geometry.

NHsp2sp3

Secondary amine with a hybrid sp2/2p3 geometry.

NH2

Neutral amine in aniline analog (pyramidal, six H2 orientations).

NH3

Ionized primary amine, as in lysine (rotates).

SCH3

Methyl attached to a sulfur, as in methionine (low rotational barrier, high number of orientations sampled).

Water

Water or hydrogen sulfide, a water analog (each can have many orientation states). Waters are not included in mover clusters to avoid combinatorial explosion, instead they are optimized after other movers.

See also

- *Adding Hydrogens and Optimizing Hydrogen Bonds* section
- `OEPlaceHydrogens` function
- *OEPlaceHydrogensDetails* class
- *OEPlaceHydrogensMoverNote* class

OEPlaceHydrogensWaterProcessing

This namespace contains constants used by the `SetWaterProcessing` and `GetWaterProcessing` methods of *OEPlaceHydrogensOptions* to control how waters (and H2S molecules) are processed by `OEPlaceHydrogens`.

Default

Same as *FullSearch*.

Focused

Sample water orientations with respect to nearby donors and acceptors (in all combinations: A, D, A/A, D/D, A/D).

FullSearch

Same as *Focused* but with two additional equilibration passes over the waters.

Ignore

Do not sample more than one orientation for each water molecule.

See also

- *Adding Hydrogens and Optimizing Hydrogen Bonds* section
- *OEPlaceHydrogensOptions.SetWaterProcessing* method
- *OEPlaceHydrogens* function

OEProteinConstraintType

Constants in this namespace are used by the *OEReceptorProteinConstraint* class to specify the type of ligand atom that is allowed to satisfy a protein docking constraint.

Chelator

A chelating heavy atom on the ligand must interact with the protein metal constraint atom to satisfy the constraint.

Acceptor

A hydrogen bond acceptor on the ligand must interact with the protein donor to satisfy the constraint.

Donor

A hydrogen bond donor on the ligand must interact with the protein acceptor to satisfy the constraint.

Contact

A heavy atom on the ligand must contact the protein constraint atom to satisfy the constraint.

Lipophilic

A non-polar heavy atom on the ligand must contact the protein

Unknown

Unknown constraint type. Most likely this receptor was written by a future version of this toolkit or FRED.

OEProtTorType

This namespace contains constants for specifying torsion angles.

Unknown

Unknown torsion.

Phi

The protein backbone ϕ dihedral angle between C(n-1)–N–CA–C(O) atoms.

Psi

The protein backbone ψ dihedral angle between N–CA–C(O)–N(n+1) atoms.

Omega

The ω dihedral angle across the peptide bond, between CA–C(O)–N(n+1)–CA(n+1) atoms.

Chi1

The protein sidechain χ_1 dihedral angle between N–CA–CB–XG(1) atoms.

Chi2

The protein sidechain χ_2 dihedral angle between CA–CB–XG(1)–XD(1) atoms.

Chi3

The protein sidechain χ_3 dihedral angle between CB–XG–XD–XE(1) atoms.

Chi4

The protein sidechain χ_4 dihedral angle between CG–XD–XE–XZ atoms.

Chi5

The protein sidechain χ_5 dihedral angle between CD–XE–XZ–XH(1) atoms.

Max

Total number of measurable protein dihedral angles (with indices: 0 .. Max-1)

OERamaCategory

This namespace contains constants used to indicate which region of the Ramachandran plot a residue is in.

See also

- *OEGetRamachandranCategoryName* function
- *OERamachandranAnalysis* class
- *OERamaType* namespace

Favored

Residue is in the favored region.

Allowed

Residue is in the allowed region.

Outlier

Residue is an outlier.

Unknown

Data for the residue cannot be calculated.

OERamaType

This namespace contains constants for specifying Ramachandran data for different amino acid residue types.

Note

The hierarchy used to determine the residue type is: *Glycine* > *Isoleucine/Valine* > *pre-Proline* > *Cis/Trans Proline* > *General*, as defined by [Chen-2010].

See also

- *OEGetRamachandranTypeName* function
- *OERamachandranAnalysis* class
- *OERamaCategory* namespace

General

The general residue type, meaning standard residues not captured by the below types.

Glycine

The data used for the Glycine residues

CisPro

The data used for the cis-Proline residues

TransPro

The data used for the trans-Proline residues

PrePro

The data used for a residue just prior to Proline in the protein chain, except Glycine, Isoleucine and Valine

IleVal

The data used for Isoleucine and Valine residues

Unknown

Unknown residue type, or the dihedral angles used to determine the type cannot be measured, e.g. for terminal residues.

OEResidueDatabaseCategory

This namespace contains the following constants used as parameters to *OEResidueCategoryData* methods and *OEGetResidueDatabaseCategoryName*. These constants refer to lists of residue names for various components found in a macromolecular complex. One of the key uses of these lists is to identify molecules that are *not* ligands, so that ligands may be recognized more easily.

AminoAcid

Standard amino acid and common variants such as *hydroxy-proline*.

Cofactor

Cofactors, such as *heme* or *flavin adenine dinucleotide*.

CofactorAndLigand

Compounds such as *ATP* that frequently act as a cofactor in some contexts and a ligand in another.

Ligand

Ligand. If the list associated with this category is populated, only ligands with one of the listed residue names will be recognized as such.

Lipid

Lipids.

Max

Total number of categories.

Metal

Metals.

Misc

Otherwise unclassified compounds that are not to be considered ligands (such as buffers or excipients).

NucleicAcid

Nucleic acids.

Polymer

Polymer residues.

Saccharide

Sugars.

OERotamerLibrary

This namespace contains constants for selecting sidechain rotamer libraries.

See also

- *OERotamer* class
- *OEGetRotamers* function

Default

The default rotamer library type is *OERotamerLibrary.Richardson_2016*.

Dunbrack

Uses rotamers from a backbone-dependent rotamer library [Dunbrack-1997] obtained from a Bayesian statistical analysis of high-resolution crystal structures.

Richardson

Use rotamers from the "penultimate rotamer library" [Lovell-2000] obtained from high-resolution crystal structures, filtered to remove clashes and residues with high-temperature factors.

Richardson_2016

Use rotamers from the "ultimate rotamer library" [Hintze-2016] obtained using a diverse high quality crystal structure dataset with associated structure factors. The structures were filtered more stringently than the original Richardson [Lovell-2000] and quality measures were performed on a per residue basis.

OESaltBridgeInteractionHintType

This namespace contains constants for the *OESaltBridgeInteractionHint* interaction type.

See also

- *OEInteractionHint* class
- *OEIsSaltBridgeInteractionHint* functor

LigandNegative

A salt-bridge interaction with the *OESaltBridgeInteractionHintType.LigandNegative* type represents an interaction where the 'ligand' component of an active site has **negative** charge and the 'protein' component of an active site has **positive** charge.

LigandPositive

A salt-bridge interaction with the *OESaltBridgeInteractionHintType.LigandPositive* type represents an interaction where the 'ligand' component of an active site has **positive** charge and the 'protein' component of an active site has **negative** charge.

ProteinNegative

Same as the *OESaltBridgeInteractionHintType.LigandPositive* constant.

ProteinPositive

Same as the *OESaltBridgeInteractionHintType.LigandNegative* constant.

UnpairedLigand

> **Warning**
>
> This is a deprecated API. Please use *OESaltBridgeInteractionHintType.UnpairedLigandPositive* or *OESaltBridgeInteractionHintType.UnpairedLigandNegative* instead.

UnpairedLigandPositive

A **positively** charged ligand atom or group near the protein without a significant interaction with the protein.

UnpairedLigandNegative

A **negatively** charged ligand atom or group near the protein without a significant interaction with the protein.

UnpairedProtein

> **Warning**
>
> This is a deprecated API. Please use *OESaltBridgeInteractionHintType.UnpairedProteinPositive* or *OESaltBridgeInteractionHintType.UnpairedProteinNegative* instead.

UnpairedProteinPositive

A **positively** charged protein atom or group near the ligand without a significant interaction with the ligand.

UnpairedProteinNegative

A **negatively** charged protein atom or group near the ligand without a significant interaction with the ligand.

IntramolecularLigand

An intramolecular ligand saltbridge.

IntramolecularProtein

An intramolecular protein saltbridge.

OESecondaryStructure

This namespace contains the following constants used as options to *OEHasSecondaryStructure* and as return values from *OEGetSecondaryStructureType*. These constants represent sets of bits, and in some cases can be OR'ed together, for example, to designate left-handed alpha helix.

Unassigned

This indicates secondary structure which is unassigned or unknown.

Helix

This indicates any type of helix.

Helix310

This indicates a 3-10 helix.

HelixAlpha

This indicates an alpha helix.

HelixPi

This indicates a pi helix.

Turn

This indicates any type of turn.

Turn3

This indicates a 3-residue turn.

Turn4

This indicates a 4-residue turn.

Turn5

This indicates a 5-residue turn.

Sheet

This indicates a beta sheet.

StrandSense

This bit indicates a strand that is antiparallel to an adjacent strand in a beta sheet. Since strands can be simultaneously parallel and antiparallel to other strands in a sheet, this isn't an absolute indicator of a strand's direction. It is useful in some visualization programs to visually illustrate antiparallel character of beta sheets.

HelixLeft

This bit indicates left-handedness of a helix. For example, a left-handed alpha helix would have a value of HelixAlpha | HelixLeft.

Any

This will match any assigned values for secondary structure when used as an option to *OEHasSecondaryStructure*.

OESeqAlignmentMethod

This namespace contains constants for selecting protein sequence alignment distance matrices.

None

Do not use a protein alignment distance matrix.

Identity

Use an identity matrix for protein alignment distance calculation (+1 if residue type is identical, otherwise 0).

PAM250

Use the PAM250 protein alignment distance matrix [Dayhoff-1978]. This matrix uses weights derived from an estimate of evolutionary substitution rates from a database of 71 groups of closely related, mostly globular proteins, with an evolutionary distance of 250 substitutions per hundred residues. This matrix is is best used when aligning similar proteins.

BLOSUM62

Use the Blosum62 protein alignment distance matrix [Henikoff-1992]. This matrix uses weights derived from alignable regions from the BLOCS database for distantly related proteins.

GONNET

Use the Gonnet protein alignment distance matrix [Gonnet-1992]. This matrix uses weights derived from exhaustive pairwise alignments of the protein sequence databases. This matrix is useful for alignments proteins of varying similarity.

OESplitMolComplexParam

This namespace defines constants used with *OESetupSplitMolComplexOptions*.

DoNotOverride

Indicates that this parameter should *not* override the value defined by the input *OEInterface* object.

OESplitMolComplexSetup

This namespace encodes symbolic constants used as bit-masks to indicate which parameters are created when the *OEConfigureSplitMolComplexOptions* function is called to configure command-line options.

All

The combination of following constants:

- *OESplitMolComplexSetup.CovBondTreatment*
- *OESplitMolComplexSetup.CovCofactor*
- *OESplitMolComplexSetup.CovLig*
- *OESplitMolComplexSetup.LigFilter*
- *OESplitMolComplexSetup.LigName*
- *OESplitMolComplexSetup.MaxSiteDist*
- *OESplitMolComplexSetup.MaxSurfDist*
- *OESplitMolComplexSetup.ModelNum*
- *OESplitMolComplexSetup.ProtFilter*

- *OESplitMolComplexSetup.SeparateRes*
- *OESplitMolComplexSetup.SiteNum*
- *OESplitMolComplexSetup.SurfWat*
- *OESplitMolComplexSetup.WatFilter*

CovBondTreatment

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the splitting of covalent ligand bonds by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -covalentbondtreatment
    Type : string
    Allow list : false
    Default : hydrogen
    Simple : false
    Required : false
    Legal values : hydrogen rgroup none
    Brief : Covalent bond treatment
    Detail
        Add hydrogens, add Rgroups or do nothing when a covalent bond is broken
```

See also

- *OESplitMolComplexOptions.SetAdjustHCount* method
- *OESplitMolComplexOptions.SetAddRGroup* method

CovCofactor

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the treatment of covalent cofactors by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -covalentcofactor
    Aliases : -covcofact
    Type : bool
    Allow list : false
    Default : false
    Simple : false
    Required : false
    Brief : Split covalent cofactors
    Detail
        Identify and separate covalently bound cofactors
```

See also

- *OESplitMolComplexOptions.SetSplitCovalentCofactors* method

CovLig

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the treatment of covalent ligands by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -covalentligand
    Aliases : -covlig
    Type : bool
    Allow list : false
    Default : false
    Simple : true
    Required : false
    Brief : Split covalent ligands
    Detail
        Identify and separate covalently bound ligands
```

See also

- *OESplitMolComplexOptions.SetSplitCovalent* method

LigFilter

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the ligand filter used by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -ligandfilter
    Type : string
    Allow list : false
    Default : ligand
    Simple : false
    Required : false
    Legal values : protein ligand water saccharide cofactor cofactor_and_ligand
                   metal counter_ion prot_complex peptide dna_rna nucleotide
                   covalent solution misc unknown
                   underdefined site1 surface_waters1 model1 chainA anything nothing
    Brief : Ligand filter category
    Detail
        Return molecules from this category as the ligand
```

See also

- *OESplitMolComplexOptions.ResetFilters* method

- *OESplitMolComplexOptions.SetLigandFilter* method

LigName

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the identification of ligands by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -ligandname
    Aliases : -ligname
    Type : string
    Allow list : false
    Default : (parameter does not have a default)
    Simple : true
    Required : false
    Brief : Ligand name
    Detail
        Select ligand having this residue name
```

See also

- *OESplitMolComplexOptions.Constructors*

MaxSiteDist

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the identification of binding sites by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -maxsitedistance
    Type : double
    Allow list : false
    Default : 8.0
    Simple : false
    Required : false
    Legal ranges :
        9999.9 to 1.0
    Brief : Maximum distance to be associated with the binding site
```

See also

- *OESplitMolComplexOptions.SetMaxBindingSiteDist* method

MaxSurfDist

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the identification of surface waters by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -maxsurfacedistance
    Type : double
    Allow list : false
    Default : 5.0
    Simple : false
    Required : false
    Legal ranges :
        9999.9 to 1.0
    Brief : Maximum distance to be associated with the protein surface
    Detail
        Used when surface waters are selected
```

See also

- *OESplitMolComplexOptions.SetMaxSurfaceWaterDist* method

ModelNum

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the selection filters used by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -modelnum
    Type : unsigned
    Allow list : false
    Default : 1
    Simple : false
    Required : false
    Legal ranges :
        999 to 0
    Brief : Select this NMR model number
    Detail
        Molecules that are not marked as models are treated as model 1.
        Model 0 selects all models.
```

Note

It may be necessary to remove the input flavor *OEIFlavor.PDB.ENDM* to read all models (see the *splitmolcomplex*

example for how to do this). In most situations, this parameter should be set to either 0 (allow all models to be selected) or 1 (the default). The reason for this is that the site counter, which the parameter *-bindingsitenum* selects based on, does not reset for each model and therefore site and model are not independent. A good example of where this can be used is with a PDB biological assembly file: setting the *-modelnum* parameter to 0 (use all models), allows a binding site spanning symmetry related proteins marked as different models to be recognized.

See also

- *OESplitMolComplexOptions.ResetFilters* method

ProtFilter

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the protein filter used by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -proteinfilter
    Type : string
    Allow list : false
    Default : prot_complex
    Simple : false
    Required : false
    Legal values : protein ligand water saccharide cofactor cofactor_and_ligand
                   metal counter_ion prot_complex peptide dna_rna nucleotide
                   covalent solution misc unknown
                   underdefined site1 surface_waters1 model1 chainA anything nothing
    Brief : Protein filter category
    Detail
        Return molecules from this category as the protein
```

See also

- *OESplitMolComplexOptions.ResetFilters* method
- *OESplitMolComplexOptions.SetProteinFilter* method

SeparateRes

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the whether individual protein residues are separated before selection by *OESplitMolComplex* and *OEGetMolComplexComponents*. This is commonly used to return just binding-site residues.

```
Contents of parameter -separateresidues
    Type : bool
    Allow list : false
    Default : false
    Simple : true
```

(continues on next page)

(continued from previous page)

```
    Required : false
    Brief : Separate individual residues before selection
    Detail
        Used to return just those residues near the binding site
```

See also

- *OESplitMolComplexOptions.SetSeparateResidues* method

SiteNum

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the selection filters used by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -bindingsitenum
    Aliases : -sitenum
    Type : unsigned
    Allow list : false
    Default : 1
    Simple : false
    Required : false
    Legal ranges :
        999 to 0
    Brief : Select this binding site
    Detail
        Specifying site 0 means select ALL sites
```

See also

- *OESplitMolComplexOptions.ResetFilters* method

SurfWat

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the treatment of surface waters by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -surfacewaters
    Aliases : -surfwat
    Type : bool
    Allow list : false
    Default : false
    Simple : true
    Required : false
    Brief : Select surface waters
```

(continues on next page)

(continued from previous page)

```
    Detail
        Waters near the surface of the protein rather than just near the binding site
```

See also

- *OESplitMolComplexOptions.ResetFilters* method

WatFilter

Passing this constant to the *OEConfigureSplitMolComplexOptions* function results in generating the following default interface to configure the water filter used by *OESplitMolComplex* and *OEGetMolComplexComponents*.

```
Contents of parameter -waterfilter
    Type : string
    Allow list : false
    Default : water
    Simple : false
    Required : false
    Legal values : protein ligand water saccharide cofactor cofactor_and_ligand
                   metal counter_ion prot_complex peptide dna_rna nucleotide
                   covalent solution misc unknown
                   underdefined site1 surface_waters1 model1 chainA anything nothing
    Brief : Water filter category
    Detail
        Return molecules from this category as water
```

See also

- *OESplitMolComplexOptions.ResetFilters* method
- *OESplitMolComplexOptions.SetWaterFilter* method

OEStackingInteractionHintType

This namespace contains constants for the *OEStackingInteractionHint* interaction type.

See also

- *OEInteractionHint* class
- *OEStackingInteractionHint* class
- *OEIsStackingInteractionHint* functor
- *OEIsIntermolecularStackingInteractionHint* functor
- *OEIsIntramolecularStackingInteractionHint* functor

Pi

A stacking interaction with the *OEStackingInteractionHintType.Pi* type represents a Pi-stacking interaction between an aromatic ring of the 'ligand' and an aromatic ring of the 'protein'.

T

A stacking interaction with the *OEStackingInteractionHintType.T* type represents a T-stacking interaction between an aromatic ring of the 'ligand' and an aromatic ring of the 'protein'.

IntramolecularLigandPi

A stacking interaction with the *OEStackingInteractionHintType.IntramolecularLigandPi* type represents an intramolecular Pi-stacking interaction between two aromatic rings within the **ligand**.

IntramolecularLigandT

A stacking interaction with the *OEStackingInteractionHintType.IntramolecularLigandT* type represents an intramolecular T-stacking interaction between two aromatic rings within the **ligand**.

IntramolecularProteinPi

A stacking interaction with the *OEStackingInteractionHintType.IntramolecularProteinPi* type represents an intramolecular Pi-stacking interaction between two aromatic rings within the **protein**.

IntramolecularProteinT

A stacking interaction with the *OEStackingInteractionHintType.IntramolecularProteinT* type represents an intramolecular T-stacking interaction between two aromatic rings within the **protein**.

4.7.3 OEBio Functions

OEAtomMatchResidueIDToString

```
std::string OEAtomMatchResidueIDToString(const OEBio::OEAtomMatchResidueID& res)
```

Returns a human-readable string representation of the given *OEAtomMatchResidueID* object, encoding its residue name, residue number, insertion code, chain ID, fragment number, and alternate location fields.

See also

- *OEAtomMatchResidueID* class
- *OEGetAtomMatchResidueIDs* function

OEBuildUnitCell

```
bool OEBuildUnitCell(OEChem::OEMCMolBase& cell,
                     const OEChem::OEMolBase& mol)
```

Apply all symmetry operators specified in mol and generate the unit cell by removing all redundant operators. `mol` must have 3D coordinates assigned and crystal symmetry set (see *OESetCrystalSymmetry*).

OECalculateDPI

Attention

This API is currently available in C++ and Python.

```
bool OECalculateDPI(OEIridiumData& irid, const OEChem::OEMolBase& mol)
```

Returns `true` if the diffraction-component precision index (DPI) value can be calculated; otherwise, it returns `false`. The calculated DPI value, the Rfree value, and the resolution of the crystal structure are stored in the `irid` (an *OEIridiumData* object).

If the DPI is not readily readable from the PDB header data, it is calculated using the Blow equation (eq.1 in [Warren-2012]), derived from work by Blow [Blow-2002] and Cruickshank [Cruickshank-1999].

See also

- *OEIridiumData* class
- Iridium section in **OESpruce TK**

OEClearDesignUnitStyle

Attention

This API is currently available in C++ and Python.

```
void OEClearDesignUnitStyle(OEBio::OEDesignUnit& du)
```

Removes all visual style information from every component of the design unit. After this call, no display style is set on any atom or bond in any tagged component.

See also

- *OESetDefaultStyle* function
- *OEDesignUnit* class

OEClearMolComplexSDData

```
void OEClearMolComplexSDData(OEChem::OEMolBase& mol,
                             const std::string& prefix =␣
↪std::string(OEMolComplexData::SDTagPrefix))
```

Deletes any SD tags that have the specified prefix. Normally deletes all the tags generated by *OEGetMolComplexComponents*, but a subset can be deleted by providing a more specific prefix.

See also

- *OEGetMolComplexComponents* function
- *OEMolComplexData.SDTagPrefix* constant

OEClearPlaceHydrogensMetadata

```
void OEClearPlaceHydrogensMetadata(OEChem::OEMolBase& mol)
```

Clears any metadata added to the molecule by *OEPlaceHydrogens*.

See also

- *OEPlaceHydrogens* function
- *OEPlaceHydrogensOptions.SetAddMetadata* method

OECombineMolComplexFragments

```
bool OECombineMolComplexFragments(OEChem::OEMolBase& dst,
                                  const std::vector<OEChem::OEAtomBondSet>& frags,
                                  const OESplitMolComplexOptions& opt =␣
↪OESplitMolComplexOptions())
bool OECombineMolComplexFragments(OEChem::OEMolBase& dst,
                                  const std::vector<OEChem::OEAtomBondSet>& frags,
                                  const OESplitMolComplexOptions& opt,
                                  const OESystem::OEUnaryPredicate<OESystem::OERoleSet>&␣
↪filter,
                                  OEChem::OEAtomBase** atomMap = NULL,
                                  OEChem::OEBondBase** bondMap = NULL)
```

Constructs a (subset) molecule `dst` from the input *OEAtomBondSet* vector `frags`. The optional *OESplitMolComplexOptions* object controls this process. If an explicit `filter` is not provided, the filter returned by the *OESplitMolComplexOptions* method *GetCombinedFilter* method will be used. You can generate your own filters, see *Splitting Macromolecular Complexes*.

Using an *OEAtomBondSet* vector and then filtering several times to produce molecules can be much more efficient compared to using the *OESplitMolComplex* or *OEGetMolComplexComponents* functions.

Any molecules returned have been given titles. Protein chains are given a shortened version of the input title combined with the chain ID. Small molecule titles are residue names. Molecules with more than one residue are combined with a `-` if they are covalently attached, and a `.` if not. Residue names are not repeated. So, a single two residue molecule combined with any number of waters might have the title `XYZ-ABC.HOH`. Output molecules may have *SD Tags* to indicate they include a covalent ligand.

Atom and bond maps may be provided as a way to map from the original `mol` used to generate the *OEAtomBondSet* vector to the `dst` molecule. They are arrays of pointers of size *GetMaxAtomIdx* and *GetMaxBondIdx*, respectively, indexed by atom and bond indices (using *OEAtomBase.GetIdx* and *OEBondBase.GetIdx*).

Returns `false` if the task could not performed. To check if the filter identified any atoms, check the number of atoms in `dst`.

See also

- *Splitting Macromolecular Complexes* section
- *OEGetMolComplexFragments* function
- *OEAtomBondSet* class
- *OESplitMolComplexOptions* class
- *OEMolComplexFilter* class
- *OESplitMolComplexOptions.GetLigandFilter* method
- *OESplitMolComplexOptions.GetProteinFilter* method
- *OESplitMolComplexOptions.GetWaterFilter* method
- *OESplitMolComplexOptions.GetOtherFilter* method
- *OESplitMolComplex* function
- *OEGetMolComplexComponents* function

OEConfigureSplitMolComplexOptions

```
bool OEConfigureSplitMolComplexOptions(OESystem::OEInterface& itf,
                                       unsigned config = OESplitMolComplexSetup::All)
```

Configures the given interface to add command-line interface parameters that control *OESetupSplitMolComplexOptions*.

itf
The interface being configured.

config
The option that specifies which parameters will be added to the interface This value has to be from the *OESplitMolComplexSetup* namespace.

See also

- *Splitting Macromolecular Complexes* section
- *OEInterface* class in the **OEChem TK** manual

- *OESplitMolComplexSetup* namespace

OECopyCrystalSymmetry

```
bool OECopyCrystalSymmetry(OEChem::OEMolBase& dst,
                           const OEChem::OEMolBase& src)
```

Copies any existing crystal symmetry from the source molecule to the destination molecule. This function returns `false` if there was no symmetry to copy and `true` otherwise.

OECountMolComplexSites

```
unsigned OECountMolComplexSites(const OEChem::OEMolBase& mol,
                                const OESplitMolComplexOptions& opt =␣
↪OESplitMolComplexOptions())
```

Returns the number of binding sites identified in the specified *OEMolBase* by identifying any ligands and then counting the number of binding sites they define. The optional *OESplitMolComplexOptions* class can be used to control this process.

```
unsigned OECountMolComplexSites(const std::vector<OEChem::OEAtomBondSet>& frags)
```

Returns the number of binding sites as above, but in this case based on the *OEAtomBondSet* vector produced by the *OEGetMolComplexFragments* function.

See also

- *Splitting Macromolecular Complexes* section
- *OEGetMolComplexFragments* function
- *OESplitMolComplexOptions* class
- *OESplitMolComplex* function
- *OEGetMolComplexComponents* function

OEExpandCrystalSymmetry

```
bool OEExpandCrystalSymmetry(OEChem::OEMolBase& mol, float radius,
                             float* center = 0)
```

Expands atoms to fill the desired radius and optional center using the existing crystal symmetry on the molecule. If no center is specified, the center of the molecule is used. This function adds symmetry atoms and bonds to the molecule. `mol` must have 3D coordinates assigned and crystal symmetry set (see *OESetCrystalSymmetry*).

```
bool OEExpandCrystalSymmetry(OEChem::OEMCMolBase& expanded,
                             const OEChem::OEMolBase& src)
```

Apply each symmetry operator to the input molecule `src` and generate a new conformer for each operator in `expanded`. `src` must have 3D coordinates assigned and crystal symmetry set (see *OESetCrystalSymmetry*).

```
bool OEExpandCrystalSymmetry(OEChem::OEMCMolBase& expanded,
                             const OEChem::OEMolBase& src,
                             float radius, float* center = 0)
```

This expands each symmetry operator within the symmetry of `src` within the given radius and center. Each symmetry operator forms a new conformer in `expanded`. `expanded` is cleared before new conformers are added.

If the center is not supplied, the center of the original molecule is used. `src` must have 3D coordinates assigned and crystal symmetry set (see *OESetCrystalSymmetry*).

OEFilterMolComplexFragments

```
OESystem::OEIterBase<OEChem::OEMolBase>* OEFilterMolComplexFragments(const std::vector
↪<OEChem::OEAtomBondSet>& frags,
                                                                     const␣
↪OESplitMolComplexOptions& opt = OESplitMolComplexOptions())
OESystem::OEIterBase<OEChem::OEMolBase>* OEFilterMolComplexFragments(const std::vector
↪<OEChem::OEAtomBondSet>& frags,
                                                                     const␣
↪OESplitMolComplexOptions& opt,
                                                                     const␣
↪OESystem::OEUnaryPredicate<OESystem::OERoleSet>& filter)
```

Returns an iterator of connected components within the input *OEAtomBondSet* vector `frags`. The optional *OESplitMolComplexOptions* object controls this process. If an explicit `filter` is not provided, the filter returned by the *OESplitMolComplexOptions* method *GetCombinedFilter* method will be used.

Any molecules returned in the iterator have been given titles. Protein chains are given a shortened version of the input title combined with the chain ID. Small molecule titles are residue names. Molecules with more than one residue are combined with a `-`. A two residue molecule might have the title `XYZ-ABC`. Output molecules also have one or more *SD Tags* to describe the functional roles.

See also

- *Splitting Macromolecular Complexes* section
- *OEGetMolComplexFragments* function
- *OEAtomBondSet* class
- *OEIter* class
- *OESplitMolComplexOptions* class
- *OEMolComplexFilter* class
- *OESplitMolComplexOptions.GetCombinedFilter* method
- *OESplitMolComplexOptions.GetLigandFilter* method
- *OESplitMolComplexOptions.GetProteinFilter* method
- *OESplitMolComplexOptions.GetWaterFilter* method
- *OESplitMolComplexOptions.GetOtherFilter* method
- *OESplitMolComplex* function
- *OEGetMolComplexComponents* function

- *OECombineMolComplexFragments* function

OEGetActiveSiteInteractionHintTypes

```
OESystem::OEIterBase<const OEInteractionHintTypeBase>*␣
↪OEGetActiveSiteInteractionHintTypes()
```

Returns an iterator over all built-in interaction types that are used to indicate protein-ligand interactions perceived by either by the *OEPerceiveInteractionHints* function or by the `OEAddDockingInteractions` function.

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Accessing Interaction Hint Information **OpenEye Python Cookbook** recipe

See also

- *OEInteractionHintTypeBase* base class
- *OECationPiInteractionHint* class
- *OEChelatorInteractionHint* class
- *OEClashInteractionHint* class
- *OEContactInteractionHint* class
- *OECovalentInteractionHint* class
- *OEHalogenBondInteractionHint* class
- *OEHBondInteractionHint* class
- *OESaltBridgeInteractionHint* class
- *OEStackingInteractionHint* class

OEGetAlignmentMethodName

```
const char* OEGetAlignmentMethodName(unsigned int method)
```

Returns the string representation of a built-in sequence alignment method.

method
This value has to be from the *OESeqAlignmentMethod* namespace.

OEGetAlignment

```
OESequenceAlignment OEGetAlignment(OEChem::OEMolBase& mol1,
                                   OEChem::OEMolBase& mol2,
                                   unsigned int assume = OEAssumption::Default,
                                   unsigned int method = OESeqAlignmentMethod::PAM250,
                                   int gap = -10, int extend = -2,
                                   bool nonStdToX = false)
```

Returns the highest scored alignment of all pairwise chain alignments calculated by *OEGetAlignments*.

mol1, mol2
The two proteins of which alignment is being calculated. No 3D coordinates are required for sequence alignment; however, if the returned *OESequenceAlignment* is passed to *OERMSD*, both molecules must have 3D coordinates assigned.

assume
A bitmask that indicates whether the algorithm can assume that the given proteins have perceived residues, have PDB ordered atoms, or have bonded residues. See the *OEAssumption* namespace for more information.

method
This value has to be from the *OESeqAlignmentMethod* namespace and it defines the algorithm that is used to generate the molecule alignment.

gap, extend
The gap and extension penalty to be used in the alignment. See also Gap penalty in Wikipedia.

nonStdToX
Option to convert non-standard residues to an X, default is false.

See also

- *OEGetAlignments* function
- *OEGetSimpleAlignment* function

OEGetAlignments

```
OESystem::OEIterBase<OEBio::OESequenceAlignment>* OEGetAlignments(OEChem::OEMolBase&␣
↪mol1,
                                                                  OEChem::OEMolBase&␣
↪mol2,
                                                                  unsigned int assume =␣
↪OEAssumption::Default,
                                                                  unsigned int method =␣
↪OESeqAlignmentMethod::PAM250,
                                                                  int gap = -10,
                                                                  int extend = -2,
                                                                  bool nonStdToX = false)
```

Returns an iterator over all pairwise chain alignments of two proteins. The returned alignments are not sorted.

mol1, mol2
The two proteins of which alignment is being calculated. No 3D coordinates are required for sequence alignment; however, if the returned *OESequenceAlignment* is passed to *OERMSD*, both molecules must have 3D coordinates assigned.

assume
A bitmask that indicates whether the algorithm can assume that the given proteins have perceived residues, have PDB ordered atoms, or have bonded residues. See the *OEAssumption* namespace for more information.

method
This value has to be from the *OESeqAlignmentMethod* namespace and it defines the algorithm that is used to generate the molecule alignment.

gap, extend
The gap and extension penalty to be used in the alignment. See also Gap penalty in Wikipedia.

nonStdToX
Option to convert non-standard residues to an X, default is false.

See also

- *OEGetAlignment* function
- *OEGetSimpleAlignment* function
- *OERMSD* function

OEGetAtomMatchResidueIDs

```
std::vector<OEAtomMatchResidueID> OEGetAtomMatchResidueIDs(const OEChem::OEMolBase& mol)
```

Retrieves the following properties for each residue in the molecule:

- residue name
- residue number
- insertion code
- chain ID
- fragment number
- alternate location.

See also

- *OEResidue* class
- *Residue Data Functors* section
- *OEAtomMatchResidue* class
- *OEAtomMatchResidueID* class

OEGetCalculatedInteractionGeometries

```
OESystem::OEIterBase<const std::string>*
OESystem::OEIterBase<const std::string>* OEGetCalculatedInteractionGeometries(const␣
↪OEInteractionHintTypeBase& itype)
```

Returns an iterator over the names of the geometries generated for the given interaction hint type.

Example:

```
OEHBondInteractionHint hbondacc = new OEHBondInteractionHint(OEHBondInteractionHintType.
↪LigandAccepts);
Console.WriteLine("Calculated geometries for '" + hbondacc.GetName() + "' interaction␣
↪type:");
foreach (String geom in OEBio.OEGetCalculatedInteractionGeometries(hbondacc))
    Console.WriteLine(geom);
```

See also

- *Accessing Calculated Interaction Hint Geometries* section
- *OEInteractionHintTypeBase* class
- *OEInteractionHint.GetCalculatedGeometries* method

OEGetChis

```
OESystem::OEIterBase<unsigned>* OEGetChis(const OEHierResidue& res)
OESystem::OEIterBase<unsigned>* OEGetChis(const OEChem::OEAtomBase* atom,
                                          unsigned assume = OEAssumption::Default)
```

Returns an iterator over the torsion-type constants from *OEProtTorType* (e.g. `Chi1`, `Chi2`) for all the side-chain χ (chi) angles of a particular residue. Pass these constants to *OEGetTorsion* to retrieve the actual angle values in radians.

OEGetCrystalSymmetry

```
bool OEGetCrystalSymmetry(OECrystalSymmetryParams& params,
                          const OEChem::OEMolBase& mol)
```

Retrieves the crystal symmetry information from the molecule. The retrieved information is stored in the given *OECrystalSymmetryParams* object.

Note

Use this overloaded function in the supported wrapped languages: Python, Java and C#.

Example:

```
OECrystalSymmetryParams p = new OECrystalSymmetryParams();
if (OEBio.OEGetCrystalSymmetry(p, mol))
{
    Console.WriteLine("a=" + p.GetA() + " b=" + p.GetB() + " c=" + p.GetC());
    Console.WriteLine(" alpha=" + p.GetAlpha() + " beta=" + p.GetBeta() + " gamma=" + p.
↪GetGamma());
    Console.WriteLine(" spacegroup=" + p.GetSpaceGroup() + " z-value= " + p.GetZValue());
}
```

```
bool OEGetCrystalSymmetry(const OEChem::OEMolBase& mol, float& a,
                          float& b, float& c, float& alpha,
                          float& beta, float& gamma,
                          unsigned int& sgnumber)
bool OEGetCrystalSymmetry(const OEChem::OEMolBase& mol, float& a,
                          float& b, float& c, float& alpha,
                          float& beta, float& gamma,
                          unsigned int& sgnumber, unsigned int& zvalue)
```

Retrieves the following the crystal symmetry information from the molecule:

- standard cell lengths
- cell angles ‘alpha’, ‘beta’ and ‘gamma’ and

- spacegroup number
- z-value

See also

- *OESetCrystalSymmetry* function

OEGetDesignUnitComponentID

Attention

This API is currently available in C++ and Python.

```
unsigned OEGetDesignUnitComponentID(const std::string& componentName)
```

This function returns numeric value from the *OEDesignUnitComponents* namespace given the input `componentName` string. It will return the *OEDesignUnitComponents.Unknown* constant if the input name does not match any known component ID.

See also

- *OEGetDesignUnitComponentName* function
- *OEGetDesignUnitComponentNamesFromMask* function

OEGetDesignUnitComponentName

Attention

This API is currently available in C++ and Python.

```
std::string OEGetDesignUnitComponentName(const unsigned componentID)
```

This function returns the name of the *OEDesignUnit* component from the input `componentID`, whose value is assumed to be from the *OEDesignUnitComponents* namespace. If the input ID does not match any of the known IDs in the namespace, the function returns “unknown”.

See also

- *OEGetDesignUnitComponentID* function
- *OEGetDesignUnitComponentNamesFromMask* function

OEGetDesignUnitComponentNamesFromMask

Attention

This API is currently available in C++ and Python.

```
std::string OEGetDesignUnitComponentNamesFromMask(const unsigned mask,
                                                  const std::string& delimiter = "|")
```

This function returns the names of the *OEDesignUnit* components from the input `mask` in a concatenated form with a delimiter. The input mask is assumed to be from the *OEDesignUnitComponents* namespace, otherwise "unknown" will be returned.

See also

- *OEGetDesignUnitComponentName* function
- *OEGetDesignUnitComponentID* function

OEGetInteractions

```
OESystem::OEIterBase<const OEInteractionHint>* OEGetInteractions(const OEChem::OEMolBase&
↪ mol,
                                                                 const␣
↪OEInteractionHintContainer& container)
```

Returns an iterator over the interactions stored in the given container that are connecting any atoms of the given molecules.

```
OESystem::OEIterBase<const OEInteractionHint>* OEGetInteractions(const␣
↪OEChem::OEAtomBase* atom,
                                                                 const␣
↪OEInteractionHintContainer& container)
```

Returns an iterator over the interactions stored in the given container that are connecting the given atom.

OEGetIridiumCategoryFromName

Attention

This API is currently available in C++ and Python.

```
unsigned OEGetIridiumCategoryFromName(std::string categoryName)
```

This function returns the corresponding Iridium category value from the *OEIridiumCategory* namespace given the input `categoryName` string.

See also

- *OEIridiumCategory* namespace
- *OEGetIridiumCategoryName* inverse function

OEGetIridiumCategoryName

Attention

This API is currently available in C++ and Python.

```
std::string OEGetIridiumCategoryName(unsigned iridiumCategory)
```

This function returns the name of the Iridium category from the input `iridiumCategory`, whose value is assumed to be from the *OEIridiumCategory* namespace.

See also

- *OEIridiumCategory* namespace
- *OEGetIridiumCategoryFromName* inverse function

OEGetMolComplexComponents

```
OESystem::OEIterBase<OEChem::OEMolBase>* OEGetMolComplexComponents(const␣
↪OEChem::OEMolBase& mol,
                                                                   const␣
↪OESplitMolComplexOptions& opt = OESplitMolComplexOptions())
OESystem::OEIterBase<OEChem::OEMolBase>* OEGetMolComplexComponents(const␣
↪OEChem::OEMolBase& mol,
                                                                   const␣
↪OESplitMolComplexOptions& opt,
                                                                   const␣
↪OESystem::OEUnaryPredicate<OESystem::OERoleSet>& filter)
```

Returns an iterator of connected components within the specified *OEMolBase*. The optional *OESplitMolComplexOptions* object controls this process. If an explicit `filter` is not provided, the filter returned by the *OESplitMolComplexOptions* method *GetCombinedFilter* will be used.

Any molecules returned in the iterator have been given titles. Protein chains are given a shortened version of the input title combined with the chain ID. Small molecule titles are residue names. Molecules with more than one residue are combined with a `-`. A two residue molecule might have the title `XYZ-ABC`. Output molecules also have one or more *SD Tags* to describe the functional roles.

Warning

The input molecule should contain *PDB residue information* such as residue name, which is used during the classification process, either from a PDB format file or through *residue perception*. Molecular complexes read from files in formats that lack this information (for example *SDF*) may be split inappropriately unless residues are perceived after input.

See also

- *Splitting Macromolecular Complexes* section
- *OEIter* class
- *OESplitMolComplexOptions* class
- *OEMolComplexFilter* class
- *OESplitMolComplex* function

OEGetMolComplexFilterCategoryName

```
const char* OEGetMolComplexFilterCategoryName(unsigned category)
```

Returns the name of the specified functional role filter category from the *OEMolComplexFilterCategory* namespace.

OEGetMolComplexFragments

```
bool OEGetMolComplexFragments(std::vector<OEChem::OEAtomBondSet>& frags,
                              const OEChem::OEMolBase& mol,
                              const OESplitMolComplexOptions& opt =␣
↪OESplitMolComplexOptions())
```

Analyze the input `mol` and generate a vector of *OEAtomBondSets* that describe each connected fragment according to its functional role. The `frags` vector can be used to *count* binding sites, *combine* fragments into a molecule, or *filter* fragments by role.

Using an *OEAtomBondSet* vector and then filtering several times to produce molecules can be much more efficient compared to using the *OESplitMolComplex* or *OEGetMolComplexComponents* functions.

To create an *OEAtomBondSet* vector:

```
OEAtomBondSetVector frags = new OEAtomBondSetVector();
```

The process is controlled by an optional *OESplitMolComplexOptions* object.

Returns `false` if the task could not be performed (for example, if `mol` is empty).

Warning

The input molecule should contain *PDB residue information* such as residue name, which is used during the classification process, either from a PDB format file or through `residue perception`. Molecular complexes read from files in formats that lack this information (for example *SDF*) may be split inappropriately unless residues are perceived after input.

See also

- *Splitting Macromolecular Complexes* section
- *OEAtomBondSet* class
- *OESplitMolComplexOptions* class
- `OECombineMolComplexFragments` function
- `OEFilterMolComplexFragments` function
- `OECountMolComplexSites` function
- `OESplitMolComplex` function

OEGetPhi

```
double OEGetPhi(const OEHierResidue& res)
double OEGetPhi(const OEChem::OEAtomBase* atom,
                unsigned assume = OEAssumption::Default)
```

Returns the ϕ (phi) angle in radians for a residue specified by its *OEHierResidue* class or by an *OEAtomBase* from the residue. The molecule must have 3D coordinates assigned.

OEGetPlaceHydrogensMetadata

```
std::string OEGetPlaceHydrogensMetadata(const OEChem::OEMolBase& mol)
```

Returns any metadata added to the molecule by `OEPlaceHydrogens`. The metadata is the string generated by a call to the `OEPlaceHydrogensDetails.Describe` method (does not require an explicit *OEPlaceHydrogensDetails* to be passed in).

See also

- `OEPlaceHydrogens` function
- `OEPlaceHydrogensOptions.SetAddMetadata` method

OEGetPlaceHydrogensMoverClassName

```
std::string OEGetPlaceHydrogensMoverClassName(unsigned moverClass)
```

Returns the standard name of the `class` assigned to a "mover" as described by a *OEPlaceHydrogensMoverNote* object generated by calling `OEPlaceHydrogens` and examining the resulting *OEPlaceHydrogensDetails* object.

See also

- *OEPlaceHydrogensMoverClass* namespace
- *OEPlaceHydrogensMoverNote.GetMoverClass* method

OEGetProteinTorsionName

```
const char* OEGetProteinTorsionName(unsigned torType)
```

Takes an integer from the *OEProtTorType* namespace and returns a `const char*` version for use in output.

OEGetPsi

```
double OEGetPsi(const OEHierResidue& res)
double OEGetPsi(const OEChem::OEAtomBase* atom,
                unsigned assume = OEAssumption::Default)
```

Returns the ψ (psi) angle in radians for a residue specified by its *OEHierResidue* class or by any *OEAtomBase* from the residue. The molecule must have 3D coordinates assigned.

OEGetRamachandranCategoryName

```
std::string OEGetRamachandranCategoryName(const unsigned int category)
```

Returns the string representation of a built-in Ramachandran category.

category
This value has to be from the *OERamaCategory* namespace.

OEGetRamachandranTypeName

```
std::string OEGetRamachandranTypeName(const unsigned ramatype)
```

Returns the string representation of a built-in Ramachandran type.

ramatype
This value has to be from the *OERamaType* namespace.

OEGetResidueAtom

These two functions return an atom pointer to a specific atom in a residue. The specific atom is specified by the *OEPDBAtomName* namespace. The residue in question can either be supplied by passing an *OEResidue* object indicating the residue and the molecule of interest, or by passing any atom in the residue. In both functions, the assume argument is a bitmask that indicates whether the algorithm can assume that the molecule has perceived residues, has PDB ordered atoms, or as bonded residues.

```
const OEChem::OEAtomBase * OEGetResidueAtom(const OEChem::OEAtomBase* atom,
                                            const unsigned pdbAtomIdx,
                                            unsigned assume = OEAssumption::Default)
OEChem::OEAtomBase * OEGetResidueAtom(OEChem::OEAtomBase* atom,
                                      const unsigned pdbAtomIdx,
                                      unsigned assume = OEAssumption::Default)
```

In this function, the residue of interest is identified by passing any atom from that residue into the function, e.g. passing in the *NZ* atom and requesting the *CA* atom in the same residue.

```
const OEChem::OEAtomBase * OEGetResidueAtom(const OEChem::OEMolBase& mol,
                                            const OEChem::OEResidue& residue,
                                            const unsigned pdbAtomIdx,
                                            unsigned assume = OEAssumption::Default)
OEChem::OEAtomBase * OEGetResidueAtom(OEChem::OEMolBase& mol,
                                      const OEChem::OEResidue& residue,
                                      const unsigned pdbAtomIdx,
                                      unsigned assume = OEAssumption::Default)
```

In this function, the residue is indicated by a combination of the molecule of interest as well as any one of the *OEResidue* objects from the residue.

See also

- *OEGetResidueAtoms* function
- *OEAssumption* namespace
- *OEHierView* for an additional method for accessing the atom of a residue in a protein

OEGetResidueAtoms

These two functions generate an iterator of all of the atoms in a residue. They differ dramatically in the manner in which the residue is specified. In both functions, the assume argument is a bitmask that indicates whether the algorithm can assume that the molecule has perceived residues, has PDB ordered atoms, or as bonded residues (see the *OEAssumption* namespace for more information). Please see *OEHierView* for an additional method for accessing the atoms of a residue in a protein.

```
OESystem::OEIterBase<OEChem::OEAtomBase>* OEGetResidueAtoms(OEChem::OEAtomBase* atom,
                                                            unsigned assume =␣
↪OEAssumption::Default)
OESystem::OEIterBase<const OEChem::OEAtomBase>* OEGetResidueAtoms(const␣
↪OEChem::OEAtomBase* atom,
                                                                  unsigned assume =␣
↪OEAssumption::Default)
```

In this function, the residue of interest is identified by passing any atom from that residue into the function.

```
OESystem::OEIterBase<OEChem::OEAtomBase>* OEGetResidueAtoms(OEChem::OEMolBase& mol,
                                                            const OEChem::OEResidue&␣
↪residue,
                                                            unsigned assume =␣
↪OEAssumption::Default)
OESystem::OEIterBase<const OEChem::OEAtomBase>* OEGetResidueAtoms(const␣
↪OEChem::OEMolBase& mol,
                                                                  const␣
↪OEChem::OEResidue& residue,
                                                                  unsigned assume =␣
↪OEAssumption::Default)
```

In this function, the residue is indicated by a combination of the molecule of interest as well as any one of the *OEResidue* objects from the residue.

See also

- *OEGetResidueAtom* function

OEGetResidueDatabaseCategoryName

```
const char* OEGetResidueDatabaseCategoryName(const unsigned category)
```

Returns the name of the specified residue name list category from the *OEResidueDatabaseCategory* namespace.

OEGetResidues

```
OESystem::OEIterBase<OEChem::OEResidue>* OEGetResidues(const OEChem::OEMolBase& mol)
```

Returns an iterator containing all the unique *OEResidue* objects in the *OEMolBase* provided. Regardless of the number of atoms in a peptide amino-acid, nucleic acid base, or ligand, only one *OEResidue* will be returned for each.

See also

- *Biopolymer Residues* section

OEGetRotamers

```
OESystem::OEIterBase<OERotamer>* OEGetRotamers(const OEHierResidue& residue,
                                               unsigned int library =␣
↪OERotamerLibrary::Default)
OESystem::OEIterBase<OERotamer>* OEGetRotamers(const OEChem::OEResidue& residue,
                                               unsigned int library =␣
↪OERotamerLibrary::Default)
OESystem::OEIterBase<OERotamer>* OEGetRotamers(unsigned int residueIndex,
                                               unsigned int library =␣
↪OERotamerLibrary::Default)
```

Return an iterator of *OERotamer* objects specifying angles $\chi_1 \cdots \chi_4$ and the associated probability for each rotamer for the given residue type based on the selected rotamer library. The type of the rotamer library has to be taken from the *OERotamerLibrary* namespace.

See also

- *OERotamer* class
- *OESetRotamer* function
- *OERotamerLibrary* namespace

OEGetSecondaryStructureID

```
int OEGetSecondaryStructureID(int ssValue)
```

Returns an integer containing the serial number of a helix, sheet, or turn in a protein. The ssValue argument is typically a value returned from *OEResidue.GetSecondaryStructure*.

OEGetSecondaryStructureName

```
std::string OEGetSecondaryStructureName(int ssValue)
```

Returns a string describing the type of secondary structure, (e.g., 'Alpha Helix', 'Beta Sheet', etc.) decoded from a secondary structure value. This value typically comes from the return value of *OEResidue.GetSecondaryStructure*.

OEGetSecondaryStructureStrandID

```
int OEGetSecondaryStructureStrandID(int ssValue)
```

Returns an integer containing the strand serial number for a strand in a beta sheet. The ssValue argument is typically a value returned from *OEResidue.GetSecondaryStructure*. For secondary structure elements other than sheets, the strand ID is always zero.

OEGetSecondaryStructureType

```
int OEGetSecondaryStructureType(int ssValue)
```

Returns an integer containing a secondary structure type decoded from the ssValue argument, which typically is returned from *OEResidue.GetSecondaryStructure*. The return values are defined in the *OESecondaryStructure* namespace.

OEGetSimpleAlignment

```
OESequenceAlignment OEGetSimpleAlignment(OEChem::OEMolBase& mol1,
                                         OEChem::OEMolBase& mol2,
                                         unsigned int assume = OEAssumption::Default,
                                         unsigned int method =␣
↪OESeqAlignmentMethod::PAM250,
                                         int gap = -10,
                                         int extend = -2,
                                         bool nonStdToX = false)
```

Returns the alignment of two proteins in an *OESequenceAlignment* object. Only the first chain of each protein is considered.

mol1, mol2
: The two proteins of which alignment is being calculated. No 3D coordinates are required for sequence alignment; however, if the returned *OESequenceAlignment* is passed to *OERMSD*, both molecules must have 3D coordinates assigned.

assume
: A bitmask that indicates whether the algorithm can assume that the given proteins have perceived residues, have PDB ordered atoms, or have bonded residues. See the *OEAssumption* namespace for more information.

method
: This value has to be from the *OESeqAlignmentMethod* namespace and it defines the algorithm that is used to generate the molecule alignment.

gap, extend
: The gap and extension penalty to be used in the alignment. See also Gap penalty in Wikipedia.

nonStdToX
: Option to convert non-standard residues to an X, default is false.

See also

- *OEGetAlignment* function
- *OEGetAlignments* function

OEGetSpaceGroupNumber

```
int OEGetSpaceGroupNumber(const std::string& ccp4Name)
```

Returns the CCP4 integer spacegroup number for the CCP4 spacegroup referenced by spacegroup. If the spacegroup cannot be found, this function returns `0`.

OEGetSpaceGroup

```
bool OEGetSpaceGroup(std::string& ccp4Name, unsigned int sgnumber,
                     bool shortname = false)
```

Assigns the standard CCP4 spacegroup name referenced by sgnumber to spacegroup. If 'shortname' is `true` then the standard shortened name for the spacegroup is assigned. This function returns `false` if 'sgnumber' does not reference a valid space group.

OEGetTorsion

```
double OEGetTorsion(const OEHierResidue& res, unsigned torType)
double OEGetTorsion(const OEChem::OEAtomBase* atom, unsigned torType,
                    unsigned assume = OEAssumption::Default)
```

Returns the angle in radians for one of the standard torsion angles in a residue. The specific torsion is indicated by passing one of the constants from the *OEProtTorType* namespace into the 'torType' parameter. The residue is specified by its *OEHierResidue* class or by any *OEAtomBase* from the residue. If the specified torsion type does not exist in the given residue, the function returns `-100.0`. The function works by calling *OEGetTorsion* with the appropriate atoms provided by the *OEProtTorType* namespace. The molecule must have 3D coordinates assigned.

See also

- *OEGetTorsion* **OEChem TK** function

OEHasBondedResidues

```
bool OEHasBondedResidues(OEChem::OEMolBase& mol,
                         unsigned assume = OEAssumption::Default)
```

Returns a boolean value indicating whether all of the atoms of each residue in the molecule are bonded to one another. The assume argument is a bitmask that indicates whether the algorithm can assume that the molecule has perceived residues, has PDB ordered atoms, or as bonded residues (see the *OEAssumption* namespace for more information).

OEHasCrystalSymmetry

```
bool OEHasCrystalSymmetry(const OEChem::OEMolBase& mol)
```

Returns a boolean value indicating whether the molecule has crystal symmetry or not.

OEHasPlaceHydrogensMetadata

```
bool OEHasPlaceHydrogensMetadata(const OEChem::OEMolBase& mol)
```

Returns `true` if metadata has been added to the molecule by *OEPlaceHydrogens*.

See also

- *OEPlaceHydrogens* function
- *OEPlaceHydrogensOptions.SetAddMetadata* method

OEIsReadableDesignUnit

Attention

This API is currently available in C++ and Python.

```
bool OEIsReadableDesignUnit(const std::string& filename)
```

Returns `true` if the supplied file name is readable by *OEReadDesignUnit*.

See also

- *OEDesignUnit* class
- *OEReadDesignUnit* function

OEIsStandardNucleicAcidResidue

```
bool OEIsStandardNucleicAcidResidue(const OEHierResidue& res)
bool OEIsStandardNucleicAcidResidue(const OEChem::OEResidue& res)
bool OEIsStandardNucleicAcidResidue(unsigned int resIdx)
bool OEIsStandardNucleicAcidResidue(const OEChem::OEAtomBase& atom)
bool OEIsStandardNucleicAcidResidue(const OESystem::OEIter<OEHierResidue>& hit)
```

Returns whether the residue referred to is one of: A, C, G, I, T, U, DA, DC, DG, DI, DT, DU Residue indices are provided by *OEGetResidueIndex*.

See also

- *OEIsStandardNucleicAcid* class for use as a predicate

OEIsStandardProteinResidue

```
bool OEIsStandardProteinResidue(const OEHierResidue& res)
bool OEIsStandardProteinResidue(const OEChem::OEResidue& res)
bool OEIsStandardProteinResidue(unsigned int resIdx)
bool OEIsStandardProteinResidue(const OEChem::OEAtomBase& atom)
bool OEIsStandardProteinResidue(const OESystem::OEIter<OEHierResidue>& hit)
```

Returns whether the residue referred to is one of: ALA, ARG, ASN, ASP, CYS, GLN, GLU, GLY, HIS, ILE, LEU, LYS, MET, PHE, PRO, SER, THR, TRP, TYR, VAL, ASX, GLX, CYX, CYH, HID, HIE or HIP. Residue indices are provided by *OEGetResidueIndex*.

See also

- *OEIsStandardAminoAcid* class for use as a predicate

OEIsTypicalPrimaryLocationCode

```
bool OEIsTypicalPrimaryLocationCode(char confCode)
```

Returns a boolean value indicating whether the alternate location code is one of the values typically used to indicate the highest occupancy conformation ('A', 'a', or '1') or is blank.

OEIsValidActiveSite

```
bool OEIsValidActiveSite(const OEInteractionHintContainer& asite)
```

Returns true if the interaction container contains exactly two molecules. One typed as "ligand" (*OELigandInteractionHintComponent*), the other as "protein" (*OEProteinInteractionHintComponent*).

See also

- *OELigandInteractionHintComponent* class
- *OEProteinInteractionHintComponent* class

Note

Valid active sites can be depicted with the **Grapheme TK**

See also

- `OEAddDockingInteractions` function in the **OEDocking TK** manual
- `OEPrepareActiveSiteDepiction` and `OERenderActiveSite` functions in the **Grapheme TK** manual

OEIsWriteableDesignUnit

Attention

This API is currently available in C++ and Python.

```
bool OEIsWriteableDesignUnit(const std::string& filename)
```

Returns `true` if the supplied file name is writable by *`OEWriteDesignUnit`*.

See also

- *OEDesignUnit* class
- *`OEWriteDesignUnit`* function

OEMolComplexChainRoleFactory

```
OESystem::OERole OEMolComplexChainRoleFactory(char chain)
```

Returns an *OERole* object corresponding to the specified chain ID for use in constructing a functional role filter.

See also

- *Splitting Macromolecular Complexes* section
- *OERole* class
- *OEMolComplexFilter* class
- *`OERoleMolComplexFilterFactory`* function

OEMolComplexFilterFactory

```
OEMolComplexFilter OEMolComplexFilterFactory(unsigned category)
```

Returns a *OEMolComplexFilter* object corresponding to the specified functional role filter category from the *`OEMolComplexFilterCategory`* namespace.

See also

- *Splitting Macromolecular Complexes* section
- *OESplitMolComplexOptions* class
- *OEMolComplexFilterCategory* namespace
- *OEMolComplexFilter* class
- *OERoleMolComplexFilterFactory* function

OEMolComplexModelRoleFactory

```
OESystem::OERole OEMolComplexModelRoleFactory(unsigned model)
```

Returns an *OERole* object corresponding to the specified model number for use in constructing a functional role filter.

See also

- *Splitting Macromolecular Complexes* section
- *OERole* class
- *OEMolComplexFilter* class
- *OERoleMolComplexFilterFactory* function

OEMolComplexOrderRoleFactory

```
OESystem::OERole OEMolComplexOrderRoleFactory(unsigned order)
```

Returns an *OERole* object corresponding to the specified component order sequence number for use in constructing a functional role filter.

See also

- *Splitting Macromolecular Complexes* section
- *OERole* class
- *OEMolComplexFilter* class
- *OERoleMolComplexFilterFactory* function

OEMolComplexRoleFactory

```
OESystem::OERole OEMolComplexRoleFactory(unsigned category)
```

Returns a *OERole* object corresponding to the specified functional role filter category from the *OEMolComplexFilterCategory* namespace.

See also

- *Splitting Macromolecular Complexes* section
- *OEMolComplexFilterCategory* namespace
- *OERole* class
- *OERoleMolComplexFilterFactory* function

OEMolComplexSiteRoleFactory

```
OESystem::OERole OEMolComplexSiteRoleFactory(unsigned site)
```

Returns an *OERole* object corresponding to the specified binding site number for use in constructing a functional role filter.

See also

- *Splitting Macromolecular Complexes* section
- *OERole* class
- *OEMolComplexFilter* class
- *OERoleMolComplexFilterFactory* function

OEMolComplexSurfaceWatersRoleFactory

```
OESystem::OERole OEMolComplexSurfaceWatersRoleFactory(unsigned site)
```

Returns an *OERole* object corresponding to the specified binding site number for use in constructing a functional role filter for surface waters.

See also

- *Splitting Macromolecular Complexes* section
- *OERole* class
- *OEMolComplexFilter* class
- *OERoleMolComplexFilterFactory* function

OEPerceiveInteractionHints

```
bool OEPerceiveInteractionHints(OEBio::OEInteractionHintContainer& asite)
```

Perceives potential interaction types between a protein and a ligand stored in the *OEInteractionHintContainer* object using default parameters.

```
bool OEPerceiveInteractionHints(OEBio::OEInteractionHintContainer& asite,
                                const OEPerceiveInteractionOptions& opts)
```

Perceives potential interaction types between a protein and a ligand stored in the *OEInteractionHintContainer* object using the given parameters (*OEPerceiveInteractionOptions*). The molecules stored in `asite` must have 3D coordinates assigned and should also have radii assigned to the atoms.

See also

- [Kumar-2002]
- [Cavallo-2016]
- [Bissantz-2010]
- [Marcou-2007]

The following interaction types are currently perceived:

- *OECationPiInteractionHint* (see also *OECationPiInteractionHintType* namespace)
- *OEChelatorInteractionHint* (see also *OEChelatorInteractionHintType* namespace)
- *OEClashInteractionHint*
- *OEContactInteractionHint*
- *OECovalentInteractionHint*
- *OEHalogenBondInteractionHint* (see also *OEHalogenBondInteractionHintType* namespace)
- *OEHBondInteractionHint* (see also *OEHBondInteractionHintType* namespace)
- *OESaltBridgeInteractionHint* (see also *OESaltBridgeInteractionHintType* namespace)
- *OEStackingInteractionHint* (see also *OEStackingInteractionHintType* namespace)

Code Examples

- *Perceive and Print Protein-Ligand Interactions* **OEChem TK** example
- Depicting Active Site Interactions **Grapheme TK** example
- Accessing Interaction Hint Information **OpenEye Python Cookbook** recipe

OEPerceiveSecondaryStructure

```
bool OEPerceiveSecondaryStructure(OEChem::OEMolBase& mol,
                                  bool force = false)
```

Calculates secondary structure for a protein molecule and stores the values into the *OEResidue* objects associated with each atom in the protein. Returns false if no secondary structure assignments were made (e.g., when passed a non-protein molecule). `mol` must have 3D coordinates assigned.

OEPlaceHydrogens

```
bool OEPlaceHydrogens(OEChem::OEMolBase& mol,
                      const OEPlaceHydrogensOptions& opts = OEPlaceHydrogensOptions())
bool OEPlaceHydrogens(OEChem::OEMolBase& mol,
                      OEPlaceHydrogensDetails& details,
                      const OEPlaceHydrogensOptions& opts = OEPlaceHydrogensOptions())
```

Function to modify the input molecule, adding explicit hydrogens oriented so as to make hydrogen bonds and, if required, "flipping" certain functional groups (e.g. imidazoles and terminal amides) that are occasionally misoriented in crystal structures.

If a *details* object is passed in, it will be updated with information that describes the results.

mol
: The molecule to add hydrogens to. Must have 3D coordinates assigned.

Warning

Just like *OEAddExplicitHydrogens*, this function can add hydrogens at the end of the input `mol`. If you require your molecule to be in *PDB order*, you may need to call *OEPDBOrderAtoms* afterwards.

See also

- *Adding Hydrogens and Optimizing Hydrogen Bonds* section
- *OEPlaceHydrogensOptions* class
- *OEPlaceHydrogensDetails* class

OEReadDesignUnit

Attention

This API is currently available in C++ and Python.

```
bool OEReadDesignUnit(const std::string& fname, OEDesignUnit& du)
bool OEReadDesignUnit(OEPlatform::oeistream& ifs, OEDesignUnit& du,
                      unsigned type = OEDesignUnitFileType::OEDesignUnit)
```

Read the contents of the file named `fname` into an *OEDesignUnit* named `du`. Returns `true` if an OEDesignUnit was read successfully, and `false` otherwise.

An overload is provided to take any arbitrary *oeistream*.

See also

- *OEDesignUnit* class
- *OEIsReadableDesignUnit* function
- *OEWriteDesignUnit* function

OERMSD

```
double OERMSD(const OEChem::OEMolBase& ref,
              const OEChem::OEMolBase& fit,
              OESequenceAlignment& align, bool onlyCAlpha = true,
              bool overlay = false, double* rot = 0, double* trans = 0)
```

This OEBio *OERMSD* function is quite analogous to the heavily overloaded OEChem *OERMSD* functions. This function takes two molecules as const parameters (one the reference and the other to be fit) and adds an additional *OESequenceAlignment* class parameter that indicates the correspondence between the two molecules. Whether the corresponding alignment considers all backbone atoms or only alpha carbons is controlled by the 'onlyCAlpha' parameter. The 'overlay' parameter indicates whether the RMSD should be measured using the current coordinates of the two molecules (default) or whether an optimum possible RMSD between the two molecules should be calculated. If the optimum overlay is calculated, the transformations applied to the 'fit' molecule to generate this optimum overlay can be retrieved by passing an array of `double[9]` to the 'rot' parameter and an array of `double[3]` to the 'trans' parameter. These two arrays can subsequently be applied to the 'fit' molecule using the *OERotate* and *OETranslate* functions if desired. It is important that *OERotate* and *OETranslate* are applied in that order.

The return value is a root-mean square distance between the specified atoms in the two molecules.

ref
: The reference molecule. Must have 3D coordinates assigned.

fit
: The molecule to be fitted. Must have 3D coordinates assigned.

OERoleMolComplexFilterFactory

```
OEMolComplexFilter OERoleMolComplexFilterFactory(const OESystem::OERole& r)
```

Returns a *OEMolComplexFilter* object corresponding to the specified *OERole*.

See also

- *Splitting Macromolecular Complexes* section
- *OERole* class
- *OEMolComplexFilter* class
- *OEMolComplexChainRoleFactory* function
- *OEMolComplexModelRoleFactory* function
- *OEMolComplexOrderRoleFactory* function
- *OEMolComplexRoleFactory* function
- *OEMolComplexSiteRoleFactory* function
- *OEMolComplexSurfaceWatersRoleFactory* function

OESecondaryStructurePacked

```
int OESecondaryStructurePacked(int ssType, int ssIDNumber,
                               int strandID = 0)
```

This function packs secondary structure information into an integer suitable for use with the *OEResidue.SetSecondaryStructure* method. The 'ssType' argument specifies a secondary structure type from the *OESecondaryStructure* namespace. 'ssIDNumber' is an ID number for a particular secondary structure element (e.g., helix #2, sheet #7, etc.), and 'strandID' is an ID number for individual strands within a beta sheet. The packed integer return value can also be used as an argument to the *OEHasSecondaryStructure* predicate, to allow identification of all atoms in a specific helix, beta sheet, turn, or strand within a sheet.

OESetCrystalSymmetry

```
bool OESetCrystalSymmetry(OEChem::OEMolBase& mol,
                          const OECrystalSymmetryParams& params,
                          bool force = false)
```

Sets the crystal symmetry stored in a *OECrystalSymmetryParams* object for a given molecule.

The above functions return `false` if the space group number (returned by *OECrystalSymmetryParams.GetSpaceGroup*) is not found or the symmetry cannot be set on the molecule because the constraints on the spacegroup are violated.

If `force` is set to true, then parameters that violate the constraints of the given spacegroup will be applied. (This is sometimes necessary based on slight rounding errors for deposited CRYST1 cards).

Note

Use this overloaded function in the supported wrapped languages: Python, Java and C#.

Example:

```
OECrystalSymmetryParams p = new OECrystalSymmetryParams(124.978f, 124.978f, 69.000f, 90.
↪0f, 90.0f, 120.0f, 154, 4);
if (OEBio.OESetCrystalSymmetry(mol, p))
{
    Console.WriteLine("a=" + p.GetA() + " b=" + p.GetB() + " c=" + p.GetC());
    Console.WriteLine(" alpha=" + p.GetAlpha() + " beta=" + p.GetBeta() + " gamma=" + p.
↪GetGamma());
    Console.WriteLine(" spacegroup=" + p.GetSpaceGroup() + " z-value= " + p.GetZValue());
}
```

```
bool OESetCrystalSymmetry(OEChem::OEMolBase& mol, float a, float b,
                          float c, float alpha, float beta,
                          float gamma, unsigned int sgnumber,
                          bool force = false)
```

Sets the crystal symmetry for a given molecule.

The above functions return `false` if the indicated space group number is not found or the symmetry cannot be set on the molecule because the constraints on the spacegroup are violated, such as a=b for tetragonal spacegroups.

If `force` is set to true, then parameters that violate the constraints of the given spacegroup will be applied. (This is sometimes necessary based on slight rounding errors for deposited CRYST1 cards).

See also

- *OEGetCrystalSymmetry* function

OESetDefaultStyle

Attention

This API is currently available in C++ and Python.

```
void OESetDefaultStyle(OEDesignUnit& du)
```

Applies a default visual style to all components of the design unit. Any previously applied style is first removed (equivalent to calling *OEClearDesignUnitStyle*), then element-based coloring and component-appropriate display styles are set for each tagged component (protein, ligand, site residues, near residues, etc.).

See also

- *OEClearDesignUnitStyle* function
- *OEDesignUnit* class

OESetRotamer

```
bool OESetRotamer(OEHierResidue& residue, const OERotamer& rotamer)
bool OESetRotamer(OEAtomBase& atom, const OERotamer& rotamer)
```

Sets the χ angles for the atoms in the specified residue. The angles are defined in the *OERotamer* object while the residue is specified either by an *OEHierResidue* object or by one of the atoms of the residue (*OEAtomBase*). The molecule must have 3D coordinates assigned; side-chain atoms are moved in 3D space to match the requested rotamer conformation.

See also

- *OERotamer* class
- *OEGetRotamers* function

OESetTorsion

```
bool OESetTorsion(OEHierResidue& res, unsigned torType, double radians)
bool OESetTorsion(OEChem::OEAtomBase* atom, unsigned torType,
                  double radians,
                  unsigned assume = OEAssumption::Default)
```

Sets one of the standard torsions in a residue to a particular value in radians. The specific torsion is indicated by passing one of the constants from the *OEProtTorType* namespace into the torType parameter. The residue is specified by its *OEHierResidue* class or by any *OEAtomBase* from the residue. The molecule must have 3D coordinates assigned; atoms in the residue are moved in 3D space to achieve the requested torsion value.

OESetupSplitMolComplexOptions

```
bool OESetupSplitMolComplexOptions(OESplitMolComplexOptions& opts,
                                   const OESystem::OEInterface& itf,
                                   unsigned siteOverride =␣
↪OESplitMolComplexParam::DoNotOverride,
```

(continues on next page)

(continued from previous page)

```
                                    unsigned modelOverride =␣
↪OESplitMolComplexParam::DoNotOverride)
```

Initializes an *OESplitMolComplexOptions* object from the parameters of a given `interface`.

The parameters `siteOverride` and `modelOverride` can be used to `ResetFilters` with values other than that from the interface object. For example, by setting `siteOverride` to `0`, filters will be set up to select for *all* sites regardless what was specified on the command line and then stored in the interface. These parameters are convenient when iterating over sites and/or models in a system. By default, these parameters will *not* override the interface object.

See also

- *Splitting Macromolecular Complexes* section
- *OEInterface* class
- `OESplitMolComplexOptions.ResetFilters` method
- `OESplitMolComplexParam` namespace

OESplitMolComplex

```
bool OESplitMolComplex(OEChem::OEMolBase& lig,
                       OEChem::OEMolBase& prot,
                       OEChem::OEMolBase& waters,
                       OEChem::OEMolBase& other,
                       const OEChem::OEMolBase& mol,
                       const OESplitMolComplexOptions& opt = OESplitMolComplexOptions())
```

Separates components of the input `mol` into the four output *OEMolBase* categories `lig`, `prot`, `waters`, and `other` representing basic elements of a macro-molecular complex. The process is controlled by an optional *OESplitMolComplexOptions* object. `Other` contains everything that is not in `lig`, `prot`, or `waters`, all of which are defined by *OEMolComplexFilter* objects from *OESplitMolComplexOptions*. By default, `lig`, `prot`, and `waters` return components associated with the first binding site. The filters can be changed so that `lig`, `prot`, and `waters` return other types of components; for example, the filters could be set so that both the protein and any waters are placed in `prot`, and any cofactors are placed in `waters`.

Any molecules returned have been given titles. Protein chains are given a shortened version of the input title combined with the chain ID. Small molecule titles are residue names. Molecules with more than one residue are combined with a `-` if they are covalently attached, and a `.` if not. Residue names are not repeated. So, a single two residue molecule combined with any number of waters might have the title `XYZ-ABC.HOH`. Output molecules may have *SD Tags* to indicate they include a covalent ligand.

Returns `false` if the task could not be performed (for example, if `mol` is empty). To check if a ligand was found, check the number of atoms in `lig`.

Warning

The input molecule should contain *PDB residue information* such as residue name, which is used during the classification process, either from a PDB format file or through `residue perception`. Molecular complexes read from files in formats that lack this information (for example *SDF*) may be split inappropriately unless residues are perceived after input.

See also

- *Splitting Macromolecular Complexes* section
- *OESplitMolComplexOptions* class
- *OEMolComplexFilter* class
- *OESplitMolComplexOptions.GetLigandFilter* method
- *OESplitMolComplexOptions.GetProteinFilter* method
- *OESplitMolComplexOptions.GetWaterFilter* method
- *OESplitMolComplexOptions.GetOtherFilter* method
- *OEGetMolComplexComponents* function
- *OECombineMolComplexFragments* function
- *OEFilterMolComplexFragments* function
- *OECountMolComplexSites* function

OESubsetDesignUnit

Attention

This API is currently available in C++ and Python.

```
bool OESubsetDesignUnit(OEDesignUnit& du, const OEDesignUnit& refDu,
                        const unsigned componentMask,
                        const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred =␣
↪OEChem::OEIsTrueAtom())
```

This function generates a fully-functional *OEDesignUnit* object (`du`) as subset of another OEDesignUnit (`refDu`) object. An additional bit mask (`componentMask`) is required to specify which components should be extracted as the subset. The bit mask should be specified from constants in the *OEDesignUnitComponents* namespace. The optional predicate can be used to subset some of the components in the specified bit mask. The predicate does not have to be complete, e.g. two water molecules can be specified to subset the solvent category, but that will not subset the protein component also included in that mask. In essence the predicate is checked to see if it matches part of a design unit component and only in this case does it take effect.

Warning

If the design unit contains a receptor it will persist in the subset, except if the subset action removes a component that was part of receptor generation mask, see *OEReceptor.GetTargetMask*

```
bool OESubsetDesignUnit(OEDesignUnit& du,
                        const unsigned componentMask,
                        const OESystem::OEUnaryPredicate<OEChem::OEAtomBase>& pred =␣
↪OEChem::OEIsTrueAtom())
This function behaves similarly as the above one, but does the subsetting
on the passed design-unit removing the need for a copy.
```

See also

- *OEDesignUnit* class
- *OEReceptor* class
- *OEDesignUnitComponents* namespace
- *OEUpdateDesignUnit* function

Examples

The example below demonstrate how to use *OESubsetDesignUnit* to subset an existing *OEDesignUnit* object.

OESwapAIEResidueAtoms

```
bool OESwapAIEResidueAtoms(OEChem::OEMolBase& mol, OEHierResidue& res)
bool OESwapAIEResidueAtoms(OEChem::OEMCMolBase& mol,
                           OEHierResidue& res)
bool OESwapAIEResidueAtoms(OEChem::OEMolBase& mol,
                           OEChem::OEAtomBase* atom,
                           unsigned assume = OEAssumption::Default)
bool OESwapAIEResidueAtoms(OEChem::OEMCMolBase& mol,
                           OEChem::OEAtomBase* atom,
                           unsigned assume = OEAssumption::Default)
```

The function name *OESwapAIEResidueAtoms* stands for `Swap Ambiguous Isoelectronic Residue Atoms`. This high-level function allows a user to easily swap the residues in an atom which may not be fully specified by the crystallographic densities. The residues which are effected by this function include ASP, ASN, ASX, GLU, GLN, GLX, HIS, HIP, HID and HIE.

The 'assume' argument is a bitmask that indicates whether the algorithm can assume that the molecule has perceived residues, has PDB ordered atoms, or as bonded residues (see the *OEAssumption* namespace for more information).

OESymmetryCartesianToFractional

```
bool OESymmetryCartesianToFractional(OEChem::OEMolBase& mol)
```

Converts the Cartesian coordinates of the molecule to unit cell fractional coordinates. Returns false if the molecule does not have crystal symmetry. `mol` must have 3D Cartesian coordinates assigned and crystal symmetry set (see *OESetCrystalSymmetry*).

OESymmetryFractionalToCartesian

```
bool OESymmetryFractionalToCartesian(OEChem::OEMolBase& mol)
```

Converts the unit cell fractional coordinates of the molecule to Cartesian coordinates. Returns false if the molecule does not have crystal symmetry. `mol` must have 3D fractional coordinates assigned and crystal symmetry set (see *OESetCrystalSymmetry*).

OEWriteAlignment

```
bool OEWriteAlignment(OEPlatform::oeostream& os,
                      const OESequenceAlignment& align,
                      unsigned int width = 50)
```

Generates a version of the standard formatted text representation of sequence alignments.

os
The stream into which the sequence alignment is being written.

align
The *OESequenceAlignment* object

width
The width of the output.

OETransformDesignUnit

Attention

This API is currently available in C++ and Python.

```
void OETransformDesignUnit(OEDesignUnit& du, const double* rmat,
                           const double* trns)
void OETransformDesignUnit(OEDesignUnit& du,
                           const OEChem::OETrans& trans)
```

Transforms the OEDesignUnit with the provided transforms. The OEDesignUnit object stores the transforms applied to it, which can be helpul if the original coordiante frame is required later in a workflow.

Warning

Applying a transform to a design unit invalidates the *OEReceptor* object stored on the design unit and it will be removed as part of the transformation.

See also

- *OETrans* class
- *OEReceptor* class
- *OEDesignUnit.GetTransform* method

OEUpdateDesignUnit

Attention

This API is currently available in C++ and Python.

```
bool OEUpdateDesignUnit(OEDesignUnit& du,
                        const OEChem::OEMolBase& mol,
                        const unsigned componentId,
                        const bool retainStyle = false,
                        const int listIndex = -1)
```

This function updates an existing *OEDesignUnit* object du by setting the component given by componentId with the input molecule mol, the input molecule must be valid (not empty). The component ID should be specified from constants in the *OEDesignUnitComponents* namespace. The optional parameter retainStyle retains the style on the input molecule, otherwise undergoes standard OEDesignUnit object styling (default). If the update function is called with an empty molecule the function returns false. In the event the update operation fails due to the inability to update a specific component, the function returns false, and the input du is cleared.

Note

List components, co-factors, excipients etc. are updated by appending to the list. To modify one of them, we recommend getting all the molecule components from a list (e.g. co-facors), clearing that component in the designunit and adding the co-factors back where one or more have been modified.

Warning

Updating the content of a design unit invalidates the *OEReceptor* object stored on the design unit and it will be removed as part of the update.

See also

- *OEDesignUnit* class
- *OEDesignUnitComponents* namespace
- *OESubsetDesignUnit* function

Examples

The example below demonstrate how to use *OEUpdateDesignUnit* to update an existing *OEDesignUnit* object.

OEWriteDesignUnit

Attention

This API is currently available in C++ and Python.

```
bool OEWriteDesignUnit(const std::string& fname,
                       const OEDesignUnit& du)
bool OEWriteDesignUnit(OEPlatform::oeostream& ofs,
                       const OEDesignUnit& du)
```

Write an *OEDesignUnit* named du to an OEDesignUnit (.oedu) file fname. This will overwrite the previous file if it existed.

The overload that takes an *oeostream* can be used to write multiple *OEDesignUnits* to the same destination. Note that the format must be specified by a constant from the *OEDesignUnitFileType* namespace as the type argument.

The function returns true if the file was written successfully, and false otherwise.

See also

- *OEDesignUnit* class
- *OEIsWriteableDesignUnit* function
- *OEReadDesignUnit* function

4.8 Preliminary OEBio API

4.8.1 Preliminary OEBio Functions

OEConstructInteractionHintContainer

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEConstructInteractionHintContainer(OEInteractionHintContainer& asite,
                                         const OEChem::OEMolBase& protein,
                                         const OEChem::OEMolBase& ligand)
```

Constructs an *OEInteractionHintContainer* object that contains the interaction hints between a protein and a ligand from the serialized interaction present in the molecules. The function will return true if the interaction hints can be constructed successfully, otherwise the function will return false and a given container will be cleared.

```
bool OEConstructInteractionHintContainer(OEInteractionHintContainer& asite,
                                         const OEBio::OEDesignUnit& du)
```

Constructs an *OEInteractionHintContainer* from the serialized interaction hints stored in an *OEDesignUnit*. The function will return true if the interaction hints can be constructed successfully, otherwise the function will return false and the container will be cleared.

See also

- *OESerializeInteractionsHintContainer* function
- *OEHasInteractionsHintSerializationData* function

OEDeleteInteractionsHintSerializationData

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
void OEDeleteInteractionsHintSerializationData(OEChem::OEMolBase& mol)
```

Deletes the serialized interaction hint data from a molecule.

```
void OEDeleteInteractionsHintSerializationData(OEBio::OEDesignUnit& du)
```

Deletes the serialized interaction hint data from all molecules in the given *OEDesignUnit*.

See also

- *OESerializeInteractionsHintContainer* function

OEDeleteInteractionsHintSerializationIds

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
void OEDeleteInteractionsHintSerializationIds(OEChem::OEMolBase& mol)
```

Deletes the serialized interaction hint ids from a molecule.

```
void OEDeleteInteractionsHintSerializationIds(OEBio::OEDesignUnit& du)
```

Deletes the serialized interaction hint ids from all molecules in the given *OEDesignUnit*.

See also

- *OESetInteractionsHintSerializationIds* function
- *OESerializeInteractionsHintContainer* function

OEHasInteractionHintSerializationIds

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OEHasInteractionHintSerializationIds(const OEChem::OEMolBase& mol)
```

Returns true if the molecule has the interaction hint serialization ids set.

See also

- *OESetInteractionsHintSerializationIds* function
- *OEDeleteInteractionsHintSerializationIds* function
- *OESerializeInteractionsHintContainer* function

OEHasInteractionsHintSerializationData

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool  OEHasInteractionsHintSerializationData(const OEChem::OEMolBase& mol)
```

Returns true if the molecule has the interaction hint serialization data set.

```
bool  OEHasInteractionsHintSerializationData(const OEBio::OEDesignUnit& du)
```

Returns true if any molecule in the given *OEDesignUnit* has the interaction hint serialization data set.

See also

- *OEDeleteInteractionsHintSerializationData* function
- *OESerializeInteractionsHintContainer* function

OESerializeInteractionsHintContainer

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
bool OESerializeInteractionsHintContainer(OEChem::OEMolBase& protein,
                                          OEChem::OEMolBase& ligand,
                                          const OEInteractionHintContainer& asite)
```

Serializes the interaction hints between a protein and a ligand from a given container and stores the serialized data in the protein and ligand molecules. The molecules can then be written to an `.oeb` file and the interaction hints can be reconstructed using the *OEConstructInteractionHintContainer* function.

See also

- *OEConstructInteractionHintContainer* function

OESetInteractionsHintSerializationIds

Attention

This is a preliminary API and may be improved based on user feedback. It is currently available in C++ and Python.

```
void OESetInteractionsHintSerializationIds(OEChem::OEMolBase& mol)
```

Sets the interaction hint serialization ids on a molecule. This function is useful when serializing interactions for multiple active sites in the same protein.

```
void OESetInteractionsHintSerializationIds(OEBio::OEDesignUnit& du)
```

Sets the interaction hint serialization ids on all molecules in the given *OEDesignUnit*. This is equivalent to calling the molecule overload on each component of the design unit.

See also

- *OEHasInteractionHintSerializationIds* function
- *OEDeleteInteractionsHintSerializationIds* function
- *OESerializeInteractionsHintContainer* function

CHAPTER

FIVE

RELEASE HISTORY

5.1 OEChem TK 5.0.0

5.1.1 New Features

- The following preliminary APIs have been added for monomer, HELM, and peptide support.

  Classes:

  - *OEMonomer* — represents a monomer in a monomer set.
  - *OEMonomerData* — stores the definition of a monomer.
  - *OEMonomerSet* — a collection of monomers.
  - *OEMonomerValidationResult* — stores the result of monomer data validation.
  - *OEHelmParsingResult* — stores the result of HELM parsing.
  - *OEHelmGenerationResult* — stores the result of HELM generation.
  - *OEHelmGenerationOptions* — gives options for HELM generation.

  Predicates:

  - *OEIsInMonomerCodeSet* — filters monomers by code set.
  - *OEHasPolymerType* — filters monomers by polymer type.
  - *OEHasMonomerType* — filters monomers by monomer type.
  - *OEHasAminoAcidType* — filters monomers by amino acid type.
  - *OEIsMonomerGroup* — identifies groups that store monomer information.
  - *OEHasMonomerGroup* — identifies groups that match a monomer group by code set and code.

  Constants:

  - *OEAminoAcidType* — amino acid type constants.
  - *OEPolymerType* — polymer type constants.
  - *OEMonomerType* — monomer type constants.
  - *OEMonomerValidationReturnCode* — monomer validation return codes.
  - *OEHelmParsingReturnCode* — HELM parsing return codes.
  - *OEHelmGenerationReturnCode* — HELM generation return codes.
  - *OEGroupType.Monomer* — monomer group type constant.
  - *OEProperty.Monomer* — monomer property type constant.

Monomer data and validation functions:

- *OEIsValidMonomerData*
- *OEAminoAcidTypeToString*
- *OEPolymerTypeToString*
- *OEStringToPolymerType*
- *OEMonomerTypeToString*
- *OEStringToMonomerType*

Monomer set I/O functions:

- *OEReadMonomerSet*
- *OEWriteMonomerSet*
- *OELoadStandardMonomerSet*
- *OELoadOpenEyeMonomerSet*

Monomer perception and detection functions:

- *OEDetectMonomers*
- *OEGetPolymerType*
- *OEGetMonomerType*
- *OEGetAminoAcidType*
- *OEIsStandardMonomer*
- *OEGetStandardAnalog*
- *OECreatePeptideMonomerConnectionSmiles*
- *OEMonomerConnectionSmilesFromCXSmiles*
- *OEPerceivePeptideMonomerConnections*

HELM functions:

- *OEHelmToMol*
- *OEMolToHelm*
- *OEGetHelmChainNames*
- *OEGetHelmMonomerCodes*

Peptide perception functions:

- *OEIsPeptide*
- *OEPerceivePeptideBackbone*

- A new function, *OEHasRadii*, has been added to check if a molecule has radii assigned to all atoms. The function takes an optional boolean argument `all` (default: `true`) to specify whether to check for radii on every atom or whether to return early on the first atom without radii.
- More *OEIFlavor* constants have been added for better control of perception steps when reading mmCIF files.

5.1.2 Minor Bug Fixes

- When generating a substructure search database with *OECreateSubSearchDatabaseFile*, molecules with high graph complexity (such as https://www.rcsb.org/ligand/ZRW) are now skipped since they are unlikely to be useful for substructure searching and can cause significant performance degradation during database generation.
- The stack size of threads used in the *OECreateSubSearchDatabaseFile* function have been increased to avoid memory allocation issues when processing `.oeb` files with large number of generic data.
- An issue has been fixed in *OEGetTorsion* where residue perception was called on a const atom changing the underlying perception state of the molecule. The function now correctly returns an invalid value when the residue is not already perceived with a warning to perceive it.

5.1.3 Python-Specific Changes

- The *OEMolBase.NewGroup* method raises `TypeError` if invoked with an invalid argument type.
- The hashing function for OEResidue has been updated to work with multicharacter chain IDs.

5.2 OEBio TK

5.2.1 New Features

- The following preliminary APIs have been added to serializing and deserializing interaction hints into `.oeb` files.
  - *OEConstructInteractionHintContainer*
  - *OEDeleteInteractionsHintSerializationData*
  - *OEDeleteInteractionsHintSerializationIds*
  - *OEHasInteractionHintSerializationIds*
  - *OEHasInteractionsHintSerializationData*
  - *OESerializeInteractionsHintContainer*
  - *OESetInteractionsHintSerializationIds*
- The serialization API for interaction hints now supports *OEDesignUnit* directly. The following functions have new overloads accepting an *OEDesignUnit*:
  - *OEConstructInteractionHintContainer*
  - *OEDeleteInteractionsHintSerializationData*
  - *OEDeleteInteractionsHintSerializationIds*
  - *OEHasInteractionsHintSerializationData*
  - *OESetInteractionsHintSerializationIds*
- The *OEInteractionHintContainer* class now supports a new *constructor overload* that accepts an *OEDesignUnit*, assembling the receptor from the design unit components and the ligand (if present) automatically.
- Intramolecular interaction perception is now supported for a single component (either ligand or protein) by passing an empty molecule for the other component to the *OEInteractionHintContainer* constructor. For example, passing an empty molecule as the ligand argument will perceive only intramolecular interactions within the protein, and vice versa. This also applies when using the *OEDesignUnit* constructor overload: if a design unit has site residues but no ligand, intramolecular receptor interactions are perceived automatically.
- The following new predicates have been added for finer-grained filtering of interaction hints:

  - *OEIsClashingHBondInteractionHint* identifies clashing hydrogen bond interactions (i.e., *OEHBondInteractionHintType.ClashAcceptorAcceptor* and *OEHBondInteractionHintType.ClashDonorDonor* types).
  - *OEIsUnpairedInteractionHint* identifies all unpaired interactions, including unpaired hydrogen bond donors/acceptors and unpaired salt-bridge charges.
  - *OEIsIntermolecularSaltBridgeInteractionHint* identifies intermolecular (ligand-protein) salt-bridge interactions.
  - *OEIsIntramolecularSaltBridgeInteractionHint* identifies intramolecular salt-bridge interactions within the ligand or protein.
  - *OEIsIntermolecularStackingInteractionHint* identifies intermolecular (ligand-protein) pi- and T-stacking interactions.
  - *OEIsIntramolecularStackingInteractionHint* identifies intramolecular pi- and T-stacking interactions within the ligand or protein.
  - *OEIsIntermolecularCationPiInteractionHint* identifies intermolecular (ligand-protein) cation-pi interactions.
  - *OEIsIntramolecularCationPiInteractionHint* identifies intramolecular cation-pi interactions within the ligand or protein.
  - *OEIsIntermolecularHalogenBondInteractionHint* identifies intermolecular (ligand-protein) halogen bond interactions.
  - *OEIsIntramolecularHalogenBondInteractionHint* identifies intramolecular halogen bond interactions within the ligand or protein.

- Intramolecular stacking and cation-pi interactions are now perceived and reported. The following new type constants have been added:
  - *OEStackingInteractionHintType.IntramolecularLigandPi*
  - *OEStackingInteractionHintType.IntramolecularLigandT*
  - *OEStackingInteractionHintType.IntramolecularProteinPi*
  - *OEStackingInteractionHintType.IntramolecularProteinT*
  - *OECationPiInteractionHintType.IntramolecularLigand*
  - *OECationPiInteractionHintType.IntramolecularProtein*
- A new option has been added to *OEPerceiveInteractionOptions* to control the minimum stack offset required for a pi-stacking or T-stacking interaction to be reported:
  - *OEPerceiveInteractionOptions.SetMinStackOffset* / *OEPerceiveInteractionOptions.GetMinStackOffset*

  The stack offset is defined as $\max(|\hat{c} \cdot \hat{n}_1|, |\hat{c} \cdot \hat{n}_2|)$, where $\hat{c}$ is the unit vector between the ring centers and $\hat{n}_1$ and $\hat{n}_2$ are the ring plane normals. This filter ensures that one ring sits above or below the other (i.e., a genuinely stacked arrangement) rather than side by side in the same plane. The default value is **0.5**.
- The following existing predicates have been updated to accept optional parameters that control whether intramolecular interactions and nonideal interactions are included in their results. All parameters default to `True` to preserve backward-compatible behavior where interactions were previously returned without distinction:
  - *OEIsHBondInteractionHint* — new constructor `OEIsHBondInteractionHint(bool includeIntra = true, bool includeNonIdeal = true)`
  - *OEIsIntermolecularHBondInteractionHint* — new constructor `OEIsIntermolecularHBondInteractionHint(bool includeNonIdeal = true)`

  - *OEIsIntramolecularHBondInteractionHint* — new constructor `OEIsIntramolecularHBondInteractionHint(bool includeNonIdeal = true)`
  - *OEIsChelatorInteractionHint* — new constructor `OEIsChelatorInteractionHint(bool includeIntra = true)`
  - *OEIsSaltBridgeInteractionHint* — new constructor `OEIsSaltBridgeInteractionHint(bool includeIntra = true)`
  - *OEIsStackingInteractionHint* — new constructor `OEIsStackingInteractionHint(bool includeIntra = true)`
  - *OEIsCationPiInteractionHint* — new constructor `OEIsCationPiInteractionHint(bool includeIntra = true)`
  - *OEIsHalogenBondInteractionHint* — new constructor `OEIsHalogenBondInteractionHint(bool includeIntra = true)`

- *OEDesignUnit* now supports an additional `SetSiteResidues` overload that accepts `std::vector<OEChem::OEResidue>`. The method validates that residues are present in the target, supports ligand coverage checks through `threshold` and `addBox`, supports optional component reassignment through `rearrangeComponents`, and updates site-associated cofactors through `siteSize`.
- A new overload to the function *OEAssignRadii* that operates on *OEDesignUnit* has been added. The function takes a mask argument and radii types to assign specific radii to specified components.
- The *OEDesignUnit* now tracks the bond order of covalent bonds between components.
- The dictionaries used in *OEResidueCategoryData* have been updated. Specifically, *GCP* has been added as a cofactor. In addition, polymers were reclassified based on how often they appear as a standalone in the wwPDB, so they can be targeted as ligands, or, if they appear frequently as part of larger complexes, they can be treated as cofactors.

5.2.2 Major Bug Fixes

- An issue has been fixed where using *OEAltLocationFactory* with the *OEAltLocFactOption.TrustInputBonds* option could lead to incorrect perception of formal charges for atoms that had an atom with multiple alternative locations bonded to it instead of proper reassignment of the implicit hydrogen count first.
- The *OEIsHBondInteractionHint* predicate previously returned `true` for clashing hydrogen bond interactions (*OEHBondInteractionHintType.ClashAcceptorAcceptor* and *OEHBondInteractionHintType.ClashDonorDonor*) and for unpaired hydrogen bond interactions (*OEHBondInteractionHintType.UnpairedLigandDonor*, *OEHBondInteractionHintType.UnpairedLigandAcceptor*, *OEHBondInteractionHintType.UnpairedProteinDonor*, and *OEHBondInteractionHintType.UnpairedProteinAcceptor*). These interaction types are now **excluded** from *OEIsHBondInteractionHint*, as they represent geometrically unfavorable or unpaired contacts rather than genuine hydrogen bonds. Use the new *OEIsClashingHBondInteractionHint* predicate to retrieve clashing interactions, and the new *OEIsUnpairedInteractionHint* predicate for unpaired interactions (note: `OEIsUnpairedInteractionHint` also returns unpaired salt-bridge interactions).
- Similarly, *OEIsIntermolecularHBondInteractionHint* no longer includes clashing types, and *OEIsIntramolecularHBondInteractionHint* no longer includes unpaired types.
- The directionality of intramolecular hydrogen bond, salt bridge, and chelator interactions has been corrected:
  - Intramolecular hydrogen bond interactions now consistently report the donor as the begin fragment and the acceptor as the end fragment.
  - Intramolecular salt bridge interactions now consistently report the positive charge as the begin fragment and the negative charge as the end fragment.

  - Intramolecular chelator interactions now consistently report the metal as the begin fragment and the chelating atom as the end fragment.

- The *OEPerceiveInteractionOptions.GetMaxAtomPairDistance* has been updated to also include the MaxUnpairedDistance in the calculation of the maximum atom pair distance.

5.2.3 Minor Bug Fixes

- *OEGetActiveSiteInteractionHintTypes* no longer returns duplicate interaction hint types.
- An issue has been fixed where incorrect perception of stacking interactions would occur. The new feature mentioned above fixes this issue with an option to control the minimum stack offset required for a pi-stacking or T-stacking.
- An issue has been fixed where chelators were incorrectly perceived because of a missing angular term.
- An issue has been fixed where halogen interactions were incorrectly perceived because of a missing angular term.
- An issue has been fixed where, in the *OEIsCTerminalAtom* and *OEIsNTerminalAtom* methods, incorrect terminal atom identification could occur for nonstandard amino acid residues.

5.2.4 Python-Specific Changes

- Python wrapping of *OEInteractionHintContainer.GetMolecules* methods has been fixed.

5.3 OEChem TK 4.3.0

5.3.1 New Features

- A new function, *OESweepMDLStereoGroups*, has been added to strip nonstereo atoms from existing enhanced stereogroups, which can be used to reconcile the stereogroup information with the existing atom stereo parities.
- A new function, *OESameAlternateLocation*, has been added to determine whether two atoms have the same alternate location set. It optionally allows for one of the states to be blank or unspecified, similar to the predicate *OEHasAlternateLocation*.
- A new function, *OEClearFormalCharges*, has been added that resets all formal charges to zero.
- A new *OECIFData* object has been added for easier access, coordination, validation, and updates to CIF header data. The full list of new features related to this object can be found on the *OECIFData* API page. Data available in this object reflects what is contained in the CIF `data_` section.
- A new *OECIFCategory* object holds CIF category data and acts as an interface to interact with this header data. The full list of new features related to this object can be found on the *OECIFCategory* API page. Data available in this object reflects what is contained in CIF pair or loop sections.
- A new *OECIFOptions* object is used during CIF update functions and informs what categories and data items in the CIF header will be updated.
- The *OESetMMCIFData* function can also take an *OECIFData* object as an input to save data stored in the object on the input molecule.
- The new function *OEUpdateMMCIFData* will update the molecule's CIF header data based on the molecule's current structure.
- The new function *OESetMMCIFChemCompData* was created to override mmCIF chemical component data on the molecule's CIF header data.
- The new function *OEIsCIFHeaderCompliant* takes the RCSB's PDB Exchange Dictionary as input and checks if the CIF categories stored on the molecule are supported by the RCSB.

- A new OEOFlavor for mmCIF, *PerceiveMetadata*, can be used to perceive all CIF header metadata at write time, based on the molecule's structure.
- Improvements have been made to CIF file reads that utilize chemical component header sections.
- The *OEIsDisulfideBridgeBond* functor identifies inter-residue disulfide bridge bonds and follows the same bond identification criteria that was in the OEBio *OEIsDisulfideBond* functor.
- The new *OEIsAmideBond* functor identifies inter-residue amide bonds.
- The new *OEIsIntraResidueBond* functor identifies intra-residue bonds.

5.3.2 Major Bug Fixes

- Improvements have been made to mmCIF file writes for stability to minimize overwriting data stored in the header.
- Improved mmCIF file write stability has been added.

5.3.3 Minor Bug Fixes

- A coding error in the assignment operator for *OEMiniMol* has been fixed.
- An issue with invalid object pointers returned from *OEGetAutomorphs* has been fixed.
- Various stability improvements to CIF header writes have been made.
- An issue reading certain LINK records from PDB files has been fixed

5.4 OEBio TK 4.3.0

5.4.1 New Features

- A method, *OEDesignUnit.GetAltLoc*, has been added to the *OEDesignUnit* class that allows the user to get the alternate location identifier for the design unit.

5.4.2 Major Bug Fixes

- An issue has been fixed in *OEAltLocationFactory* so that input options are fully respected.

5.4.3 Minor Bug Fixes

- The following were removed from the polymers category in the residue database: ZYA, R99, RL2, VSC, ZU5, S89, 0I5, 0D6, 0E6, ZRA, I0Y, and K36.
- The *OEIsDisulfideBond* functor now uses the **OEChem** *OEIsDisulfideBridgeBond* functor for its implementation.
- The covalent perception distance in design unit splitting to components has been relaxed.

5.5 OEGrid 4.3.0

5.5.1 New Features

- A new function, *OEReadCIFGridFile*, has been added to read structure factor CIF (sfCIF) files into *OESkewGrid*.

5.6 OEChem TK 4.2.0

5.6.1 New Features

- If an mmCIF file with long residue names is written to PDB, it will first check whether the three-character version is unused and will use that if it is.
- A new option parameter has been added to *OEUniMolecularRxnOptions* that enables hydrogen count conversions and perceptions of the target(s) required for the specified transformation (default: `true`).

5.6.2 Major Bug Fixes

- An issue has been fixed that caused mmCIF files with long residue names to break the long residue names into multiple residues.

5.6.3 Minor Bug Fixes

- An issue has been fixed that ensures that missing mmCIF metadata relevant for Iridium classification of a structure does not cause problems.
- An issue has been fixed which ensures that LINK records are parsed only if formatted correctly.
- A new option parameter in *OEUniMolecularRxnOptions* enables calling *OEPrepareSearch* for the query and target. This call suppresses the warning messages about unsynchronized explicit or implicit hydrogens between the internal query and the target that were unable to be remedied by any user action.
- The function *OEPrepareSearch* is now called as part of the initialization in *OELibraryGen* to eliminate warning messages about explicit or implicit hydrogens and stereo perceptions on the provided starting materials.

5.6.4 Python-Specific Changes

- A couple of issues have been fixed that caused crashes using Python 3.12, related to using *OEResidue* objects in a dictionary and setting an *OEBitVector* (or classes derived from it) or an *OEHalfFloat* as generic data on a molecule.

5.7 OEBio TK 4.2.0

5.7.1 Minor Bug Fixes

- An issue has been fixed in the *OEBio_OEIsNTerminalAtom* predicate ensuring the NME capping group is not matched.
- *OEProteinConstraintType* constants are now properly wrapped in Python.

5.8 OEChem TK 4.1.1.1

5.8.1 Major bug fixes

- An issue has been fixed where `OEReadMolecule` would fail to read a molecule from an mmCIF file if an invalid LINK record was present.

5.9 OEChem TK 4.1.1.0

5.9.1 New features

- Support has been implemented for OEGroup information for *OEMiniMol* to support the return of enhanced stereogroup information from OEFastFPDatabase hits. Previously, enhanced stereogroup information on the indexed structures would be ignored during indexing and thus missed from returned hits.
- LINK records in the PDB metadata are now parsed to improve perception of covalently bound heterogens. The equivalent *_struct_conn* metadata is also parsed from mmCIF files. The distance between the listed atoms is confirmed before applying the bond.

5.9.2 Minor bug fixes

- If any parse error results from parsing a *OEFormat.CXSMILES* appendix specifically, the full appendix plus the molecule title are now returned in the molecule title to allow user inspection and corrective action.
- *OEFormat.SDF* data parsing has been modified to ensure that >-prefixed data is properly captured. Since SDF data field name lines are >-prefixed as well, the new behavior explicitly requires a blank line delimiter to be present between data entries, including null data items, per the SDF file format specification. This allows unambiguous identification of data field name header lines versus data entry lines.
- An issue has been fixed to avoid generating duplicate data tags from generic data that can result in invalid JSON output.
- Titles returned from *OEMolDatabase.GetTitle* when *OEFormat.CXSMILES* files are indexed no longer contain the CXSMILES appendix unless there is a parse error with the appendix information itself.
- An error has been fixed that sometimes resulted in missing atom parity information in output SDF format files when a nondefault *oemolistream.SetConfTest* was provided.
- A line ending parse error has been fixed for *OEFormat.FASTA* files affecting windows files.
- Propagation of perceived query aromaticity has been added for alternating single/double Kekulé forms so that alternate and equivalent s/d bond patterns are treated consistently.
- Null pointer protections have been added to some internal search matching code that could cause a crash.
- *OEFileStringTypeFlavor.Default* has been augmented to exclude file types that are not suggested for use. Accordingly, a new constant, *OEFileStringTypeFlavor.Pretty*, has been defined and is part of the new definition of *OEFileStringTypeFlavor.Default*. Removing *OEFileStringTypeFlavor.Pretty* from *OEFileStringTypeFlavor.Default* would provide the file types that include everything.
- A new type, *OEMMFFType.CION*, has been added to define a generic type for counterions.

5.10 OEPlatform TK 4.1.1

5.10.1 New features

- The ZSTD library has been updated from version `1.5.2` to `1.5.6`.

5.11 OEBio TK 4.1.1

5.11.1 Major bug fixes

- In the Iridium categorization of a design unit, a properly perceived, covalently bound ligand no longer triggers an HT to MT category transition. The rule flags atoms that are within what is considered a covalent bond distance, where a bond is not properly perceived.

- An issue has been fixed so that *ClearComponent* can be used to clear user-defined components.

5.12 OEChem TK 4.1.0

5.12.1 New features

- We are now able to read and parse the Chem Comp section of mmCIF files.
- We can now force a deterministic sorted order to the enhanced stereo information in generated CXSMILES appendices.
- New overloads of the *OESimpleAppOptions* constructors have been added to enable their usage when an output parameter is not desired.
- A new value, `OEFileStringType.BroodQMol`, has been added to `OEFileStringType` to support molecule files containing BroodQuery.
- The following preliminary API classes have been made permanent.
  - *OECreateSubSearchDatabaseOptions*
  - *OESubSearchDatabase*
  - *OESubSearchQuery*
  - *OESubSearchResult*
  - *OESubSearchScreen*
  - *OESubSearchScreenTypeBase*
  - *OEMolRecord*
  - *OEMolParameter*
  - *OEFileStringParameter*
  - *OEMolReaderParameter*
  - *OEMolWriterParameter*
  - *OEOutFileStringParameter*
  - *OEPredicateParameter*
  - *OEUnAtomPredParameter*
  - *OEUnBondPredParameter*
  - *OERefInputAppOptions*
  - *OESimpleAppOptions*
- The following preliminary API functions have been made permanent.
  - *OEReadCIFFile*
  - *OEWriteCIFFile*
  - *OECreateSubSearchDatabaseFile*
  - *OEGetSubSearchScreenType*
  - *OEIsComplementaryScreenType*
  - *OEIsValidSubSearchDatabase*
  - *OEIsValidSubSearchScreenString*

- *OEMakeSubSearchQueryScreen*
- *OEMakeSubSearchTargetScreen*
- *OESameSubSearchScreenTypes*
- *OEScreen*
- *OESubSearchStatusToName*

5.12.2 Major bug fixes

- The handling of `OESMILESFlag.ExtBonds` formatted fragments has been improved to better support explicit cis/trans parity information for the coupling bond.
- Parse multiply referenced enhanced stereogroups correctly from CXSMILES appendices.
- The mapping behavior of *atom and Rgroup atoms for both MDL and SMARTS queries is now synchronized. Rgroups and/or *atoms in queries are now equivalently supported for `OEMakeSubSearchQueryScreen`.
- Conversion of *OERecord* to CSV now writes CXSMILES instead of SMILES to ensure enhanced stereo is not dropped.

5.12.3 Minor bug fixes

- Null argument checks have been added to `OESuppressHydrogens` to avoid causing a crash.
- Any attempt to kekulize partial structures for `OEUniMolecularRxn` has been fully suppressed when specifically requested, which allows explicit nonaromatic/aromatic transformations for partial ring structures.
- `.lst` and `.list` no longer appear as viable input file types in the options in *OESimpleAppOptions* and *OERefInputAppOptions*.

5.13 OESystem TK 4.1.0

5.13.1 New features

- The following preliminary API classes have been made permanent.
  - *OERecord*
  - *OEOptions*
  - *OEBoolParameter*
  - *OEConstParameter*
  - *OEDoubleParameter*
  - *OEFloatParameter*
  - *OEIntParameter*
  - *OEMultiParameter*
  - *OEPointerParameter*
  - *OEPODParameter*
  - *OEStringParameter*
  - *OEUIntParameter*
- The following preliminary API functions have been made permanent.

- *OEConfigureItf*
- *OEConfigureOpts*
- *OEUpdateOpts*

5.14 OEMath TK 4.1.0

5.14.1 New features

- The following preliminary API classes have been made permanent.
  - *OEKernelPLS*
  - *OEMatrix*
  - *OESquareMatrix*

5.15 OEBio TK 4.1.0

5.15.1 New features

- A new overload of `OESubsetDesignUnit` has been added to allow for in-place subsetting of a design unit to avoid generating a copy.

5.15.2 Major bug fixes

- Several issues in `OEPlaceHydrogens` have been added to address:
  - improved perception and geometry for sp^3 secondary amines.
  - improved chelation rules for primary amines.
  - improved chelation rules for phosphate and borate groups.
  - improved chelation rules for sulfonamide groups.
  - improved chelation rules for hydroxamate moieties.

5.15.3 Minor bug fixes

- Null argument checks have been added to `OEGetTorsion` to avoid causing a crash.

5.16 OEGrid TK 4.1.0

5.16.1 Major bug fixes

- A problem in *OEReadMTZ* has been fixed.

5.17 OESystem TK 4.1.0

5.17.1 New features

- The following preliminary API classes have been made permanent.
  - *OERecord*
  - *OEOptions*

  - *OEBoolParameter*
  - *OEConstParameter*
  - *OEDoubleParameter*
  - *OEFloatParameter*
  - *OEIntParameter*
  - *OEMultiParameter*
  - *OEPointerParameter*
  - *OEPODParameter*
  - *OEStringParameter*
  - *OEUIntParameter*
- The following preliminary API functions have been made permanent.
  - *OEConfigureItf*
  - *OEConfigureOpts*
  - *OEUpdateOpts*

5.18 OEMath TK 4.1.0

5.18.1 New features

- The following preliminary API classes have been made permanent.
  - *OEKernelPLS*
  - *OEMatrix*
  - *OESquareMatrix*

5.19 OEChem TK 4.0.0

5.19.1 New features

- A new preliminary API that supports generation of atom–atom mapping information for an input reaction has been provided in C++ and Python. See details at `OERxnAutomapperOptions`, `OERxnAutomapper`, and `OERxnAutomapStatus`.
- A new nondefault input flavor has been added to support the conversion of implicit enhanced stereogroups from a V2000 SD format input file into explicit stereogroups. This allows subsequent export of explicit enhanced stereogroups in formats that support that information, like *SDF* and *CXSMILES*. See details at *SDF_ForceImp2ExpENHSTE*.
- The explicit chiral-off designation (`r`) in the CXSMILES appendix is now tolerated when parsing the CXSMILES appendix. This explicit designation is appearing more frequently in vendor structure sets.
- New overloads of `OESimpleAppOptions` and `OERefInputAppOptions` have been added to support applications that do not produce an output file.
- The MMCIF reader now runs on Gemmi version 0.6.2. It is able to parse and write more sections in MMCIF files.
- The default OEIFlavor for *PDB* has been changed to include *PDB_DATA* by default.

- The default OEIFlavor for *MMCIF* has been changed to not include *MMCIF_ALTLOC* by default, to match the *PDB* default behavior.
- Two new OEIFlavor defaults have been added, *PDB_SpruceDefault* and *MMCIF_SpruceDefault*, that add `ALTLOC` in addition to the default flavors for those file formats. These are being used in both the **SPRUCE** and **Make Receptor** applications.
- *OEHasAtomName* has been extended to take a new optional argument called `trimspaces` (default=false) to more flexibly match atom names.
- *OEHasChainID* has been changed to internally operate on strings instead of characters to support multicharacter chain IDs. The character-based constructor is still available.

5.19.2 Minor bug fixes

- The SD data format specifies that blank lines are the delimiters between different SD data fields. If an SD file contains blanks within one or more SD data fields, the reader would previously complain about the format error. The reader has been augmented to be more tolerant of this case, but in the process it must strip blank lines from the input SD data to ensure that the SD output does not generate a noncompliant output file. If blank lines are detected in the SD data for a molecule being exported in SD format, a new write status is returned to indicate the blank lines were removed in order to generate a compliant SD output file, and the generated output file will have the noncompliant blank line(s) removed. See details at *OEWriteMolReturnCode.InvalidSDData*.
- A minor issue that caused *OEGetTorsion* to crash if one or more of the atom arguments were null has been fixed.
- A minor issue that generated an invalid read error from V3000 SD files with nonconsecutively ordered object indices has been fixed. Generally, object indices in V3000 format files can be arbitrarily ordered.
- A defect that could result in a crash during MCS searching for targets containing single atom fragments has been fixed.
- An issue that caused *OEMolParameter* to not be properly transmitted in MPI mode has been fixed.
- An issue with the assignment operator for *OEIsBridgeHead* being incorrectly defined has been fixed.
- New overloads of *OESameResidue* and *OESameChain* that take atoms instead of residues have been added. This will prevent the need to get copies of *OEResidue* objects in wrapped languages, instead of using const references internally.

5.20 OEBio TK 4.0.0

5.20.1 New features

- Two new methods in *OEReceptor*, *GetOuterContourVolume* and *GetInnerContourVolume*, have been added as convenience functions to get the volume of the receptor grid at the set contour level.
- The function *OEUpdateDesignUnit* will not fail and return false if passed an empty molecule to update with.
- The function *OEUpdateDesignUnit* has been updated to take an optional listIndex parameter if an item in a list molecule needs to be updated (e.g., one of several cofactors).
- The *OEDesignUnit* class has been extended with a method *ClearComponent* to remove or clear a component from a design unit; this method can also take an optional listIndex to remove a specific molecule in a list component (e.g., one of several cofactors).

5.21 OESystem TK 4.0.0

5.21.1 New features

- A new overload of the *OEGetColors* function has been added that generates a custom color iterator for use in depictions by passing a vector of desired colors.

5.22 OEPlatform TK 4.0.0

Minor internal improvements have been made.

5.23 OEGrid TK 4.0.0

Minor internal improvements have been made.

5.24 OEChem TK 3.4.0

5.24.1 New features

- New preliminary APIs have been added for generating SMARTS patterns for atoms and bonds that can then be used to construct a molecule SMARTS query:
  - *OECreateAtomSmartsString*
  - *OECreateBondSmartsString*
  - *OECreateSmartsString*
- A new overload of the *OECopyMol* has been added that takes *OEMCMolBase* as an argument.
- An issue has been fixed where the input flavor *OEIFlavor.MMCIF.ALTLOC* for retaining alternate locations when reading an MMCIF file was not being accepted as a proper constant.

5.24.2 Minor bug fixes

- Reaction mapping semantics for SMARTS queries which force the query mapping to the respective reactants or products in the target have been activated. Previously, reaction SMIRKS queries would hit either reactants or products regardless of the presence of the >>. To restore the old behavior, remove the >> from query patterns.
- An issue with the *OEIsTermHeavyAtom* predicate, which did not properly account for atom type and implicit or explicit hydrogen environments of the interrogated atom, has been fixed.
- An issue in which the status of double ether bonds was not maintained when exported in V3000 format has been fixed.
- An issue in which the nondefault output flavor flags could not be used with the *OEFormat.CXSMILES* format has been fixed.
- An issue with buffer overrun for V2000 SDF parsing that could result in erroneous application of properties from prior parsed lines has been fixed.
- An issue with stale OEGroup pointers that could result in a crash for molecule copy operations has been fixed.

5.24.3 Python-specific changes

- An option has been added to the molcount.py example that allows use of an Omega-specific ContTest handling for Omega generated conformer inputs.

5.25 OESystem TK 3.3.1

Minor internal improvements have been made.

5.26 OEPlatform TK 3.3.1

Minor internal improvements have been made.

5.26.1 C++-specific changes

- The code has been changed to define the statement "NOMINMAX" to trigger only if not already set.

5.27 OEChem TK 3.3.1

5.27.1 New features

- New preliminary API class, *OEMCSFuncParameter*, has been added to provide functionality for using *OEMCSFunc* as a parameter in *OEOptions*.

5.27.2 Major bug fixes

- An uninitialized memory issue that could impact the output of correct InChI identifiers based on prior identifier generations has been fixed.
- An issue that inadvertently collected enantiomers as equivalent conformers when *OEOmegaConfTest* conformer tests are in use has been fixed. The behavior of prior releases has been restored for these conformer tests, which results in enantiomers being considered as separate conformers.
- An issue where the `OEFastaToMol` the C-terminal residue contained an aldehyde instead of a carboxylic acid has been fixed.

5.27.3 Minor bug fixes

- The ability to modify isotopic information as part of transformation activity for `OEUniMolecularRxn`, `OEGetUniMolecularRxnIter` and `OELibraryGen` has been added.

5.28 OEBio TK 3.3.1

5.28.1 New features

- The residue code dictionary used for `OESplitMolComplex` has been updated with new lipid codes.

5.28.2 Minor bug fixes

- An issue with `OEDesignUnit` where the atom order of the stored molecules in the design unit was not properly updated when components were updated has been fixed.

5.29 OESystem TK 3.3.1

5.29.1 New features

- A new record metadata hint for CXSMILES was introduced in :OEMetadata.

5.29.2 Major bug fixes

- A bug was fixed for the *XPLOR* grid file format reader.

5.29.3 Minor bug fixes

- An issue in *OEReadRecord* that could cause the reader to go into an infinite loop when given a corrupted/truncated oedb file has been fixed.

5.30 OEPlatform TK 3.3.1

Minor internal improvements have been made.

5.31 OEChem TK 3.3.0

Spring 2022

5.31.1 New features

- Support for CXSMILES has been added as a new molecule I/O format, *OEFormat.CXSMILES*.
- A new option flag, *OESMILESFlag.EnhStereo* has been added to the *OESMILESFlag* flags for controlling the output of CXSMILES enhanced stereo information.
- New methods were added to the *OEParseSmilesOptions* class to control the consideration of CXSMILES appendices during the parse. See, *GetCXSMILES* and *SetCXSMILES*.
- New methods were added to the *OEInChIOptions* class to control the SUU option during InChI generation. This will necessarily generate nonstandard InChIs when enabled. See, *GetUndefinedStereo* and *SetUndefinedStereo*.
- New flavor flags were added to *OEFileStringParameter* and *OEOutFileStringParameter* to support file type selections. See the *OEFileStringTypeFlavor* namespace.
- Overloads for *OECalcPMI* and *OEGetRadiusOfGyration* were added that accept *OEMCMolBase* and *OEConfBase* objects.

5.31.2 Minor bug fixes

- A minor bug related to uninitialized memory in *OECalcPMI* was fixed.
- An issue disallowing the use of multiple options for *OEIFlavor* support with SDF, RDF, MDL formats has been fixed.
- An issue affecting copy activities of *OEGroupBase* objects during conformer copy activities has been fixed.
- The *GetHasValue* method now returns `false` for a non-zero input argument, as appropriate.

5.32 OEBio TK 3.3.0

5.32.1 Minor bug fixes

- An issue with the assignment operator for *OEDesignUnit* has been fixed.
- An issue with automated titles for *OEDesignUnit* has been fixed.
- An issue with the default style on generated *OEDesignUnit* related to built residues has been fixed.
- The algorithm for *OEDesignUnit* creating has been improved to better detect unnatural amino acids.

5.33 OESystem TK 3.3.0

5.33.1 New features

- A new method, `AddParameterPrefix`, has been added for adding a prefix to all the command line parameter's names.
- a new method, `GetPrettyLegalValues`, has been added to get the list of the named legal values.

5.33.2 Minor bug fixes

- An issue causing list parameters to be not updated properly in `UpdateValues` has been fixed.

5.34 OEPlatform TK 3.3.0

5.34.1 General notices

- The ZSTD dependency has been updated to version 1.5.2, which should positively effect the `OEFormat.OEZ` format type in both compression level and encoding and decoding performance.
- The ZLIB dependency has been updated to version 1.2.12 with latest security improvements.

5.35 OEGrid TK 3.3.0

5.35.1 New features

- The *mrc* file format is now recongized as a CPP4 grid file type in `OEGridFileType`.

5.36 OEChem TK 3.2.0

Fall 2021

5.36.1 New features

- A new options class, *OEUniMolecularRxnOptions*, has been added to provide control arguments to the unimolecular reaction APIs *OEUniMolecularRxn* and `OEGetUniMolecularRxnIter`. The new options class adds support for enabling valence correction, as well as retrieval of altered atom information from the transform for post-processing of the transformed products.
- Small molecule crystal structures are now writeable as `CIF` files, see `OEWriteCIFFile` for details.
- Macromolecule files are now writeable as `MMCIF` files, see `OEWriteCIFFile` for details.
- `CIF` and `MMCIF` files are now readable using *oemolithread*. This enables their use as *OEMolDatabase* as well.

- The gemmi library used to read and parse `MMCIF` files has been updated to version 0.4.8.
- New overloads of constructors for the *OESimpleAppOptions* and *OERefInputAppOptions* have been added to extend their abilities to be used when no sub-options are involved for an application or when custom file extensions are required.
- New overloads of constructors for the *OEFileStringParameter* and *OEOutFileStringParameter* have been added that accepts file extensions as arguments instead of predefined constants.

5.36.2 Major bug fixes

- An issue causing missing amide (peptide) bonds when reading protein structures from `MMCIF` files has been fixed.
- An issue related to incorrect hydrogen atom stripping from stereocenters that are asymmetric by virtue of hydrogen isotope substitutions has been fixed. Isomeric `SMILES` for structures containing hydrogen isotopes on stereogenic centers will now have the atom/bond parity information correctly preserved. A minor consequence of the fix is that a small number of isomeric `SMILES` with explicit (non-isotopic) hydrogens may have `SMILES` differences due to conversion to a fully hydrogen-suppressed `SMILES` form.

5.36.3 Minor bug fixes

- An issue that could cause a crash during reads of *OEFormat.OEB* format files has been fixed. The read and write of *OEGroupBase* information in `OEB` files is now more rigorously checked for invalid objects. Existing corrupted `OEB` files will return an error status when read.
- An issue that could cause overflow in the MDL molecule reader has been fixed.
- An issue that caused return of an incorrect default value for *OEPredicateParameter* when it was set to a non-default value has been fixed.

5.37 OEBio TK 3.2.0

5.37.1 New features

- The logic identifying aromatic nitrogen atoms has been updated in *OEPlaceHydrogens*, enabling improved hydrogen network optimization as well as the ability to properly distinguish and pick appropriate ligand tautomers in the protein binding site.

5.37.2 Minor bug fixes

- An issue causing the ligand density coverage (LaD) to change as a result of hydrogen placement in limited circumstances has been fixed.
- An issue causing the operator bool on *OEAtomMatchResidue* to return true even if no valid data was provided to the constructor has been fixed.
- An issue causing an *OEDesignUnit* attached as generic data on molecules to be lost during I/O has been fixed.
- An issue causing proper backbone verification to fail for capping groups has been fixed.
- An issue causing proper backbone verification to fail for nonstandard amino acids has been fixed.
- An issue causing histidine residues to be in the charged state (HIS+ or HIP) when it clearly was not favorable has been fixed.
- The AspDyad mover rule in *OEPlaceHydrogens* has been refined.
- The class *OEPlaceHydrogensOptions* has been modified to derive from *OEOptions*.

5.38 OESystem TK 3.2.0

5.38.1 New features

- *OEParameter* classes that accept strings as input values now work with case-insensitive string inputs.
- A new overload of `OEMultiParameter.SetValue` has been added that accepts *oeifstream* as input.

5.38.2 Minor bug fixes

- An issue that could cause `OEStringToNumber` to crash has been fixed.

5.39 OEPlatform TK 3.2.0

- Minor internal improvements have been made.

5.40 OEGrid TK 3.2.0

- Minor internal improvements have been made.

5.41 OEMath TK 3.2.0

5.41.1 New features

- A new preliminary API class, *OEKernelPLS*, has been added that builds QSAR models from kernelized descriptors using partial least squares (PLS).

5.42 OEChem TK 3.1.1

July 2021

5.42.1 New features

- OEZ has been made the default file format for storing molecules in *OERecord* objects.
- Two new functions, `OECalcPMI` and `OEGetRadiusOfGyration`, have been added. The first calculates the principal moment of inertia (PMI) for a molecule, and the second calculates the radius of gyration for that molecule using a previously calculated PMI.
- InChI has been updated to `v1.06`. A new option to *OEInChIOptions* supports generation of nonstandard InChI strings with pseudoatoms present.
- Atom/bond SMARTS expressions can now be used as generic data for SMARTS queries.
- The function `OEApplyStateFromRef` now transfers stereo/chiral information from the reference state molecule.

5.42.2 Major bug fixes

- The function `OEPerceiveResidues` has been enhanced to properly recognize ACE (acetyl) and NME (N-methyl) capping groups, which used to cause unexpected behaviors. Recognition of the NHE and FOR capping groups has also been improved.

5.42.3 Minor bug fixes

- An issue that caused a crash in the *OEPDBOrderAtoms* function when running with preserveResOrder=True and molecules that do not have chain IDs has been fixed.
- An issue that caused *OEApplyStateFromRef* to return redundant matches when doing partial matching with long hydrocarbon chains has been fixed.
- An issue that caused *OEDetermineConnectivity* to break all bonds between atoms with wrong valences (possibly resulting in a broken backbone) has been fixed. It now tries to break the bonds between non-adjacent residues first and, if the valences of involved atoms are still wrong, all bonds.

5.42.4 Documentation changes

- Added documentation for the functions *OEGetSoleNeighbor*, *OEGetSoleBond*, *OEGetSoleDoubleBond*, and *OEGetSoleSingleBond*.

5.43 OEBio TK 3.1.1

5.43.1 New features

- Two new constructors have been added to the *OEDesignUnit* class. These constructors are intended to be used with prepared molecules. The two APIs are for liganded or apo-structures, where in the latter case site-residues indicating the binding site are required.
- Predicate versions of *OEIsStandardProteinResidue* and *OEIsStandardNucleicAcidResidue* have been added as *OEIsStandardAminoAcid* and *OEIsStandardNucleicAcid*, respectively.
- The *OEReceptor* class now stores a predicate that can be used to subset an *OEDesignUnit* class component. For example, a specific water molecule in the solvent component that should be used when generating the receptor grid and during docking.
- The function *OESubsetDesignUnit* and the method *OEDesignUnit.GetComponents* now take a predicate that will subset the component(s) they hit in the design unit.
- The functions *OESubsetDesignUnit* and *OEUpdateDesignUnit* will now clear the stored OEReceptor if the targetMask and targetPredicate of the receptor match the content being updated or subsetted away.
- Non-const methods for *GetCustomConstraints*, *OEReceptor.GetProteinConstraints*, and *GetConstraintFeatures* have been added giving access to edit constraint features.

5.43.2 Minor bug fixes

- Nitriles (C#N) are now recognized as hydrogen bond acceptors in interaction hint perception.
- Hydrogen bonds to moeities that are also interacting as saltbridges are now properly perceived.
- Amide NH atoms are now properly counted as as UnpairedLigandDonor when appropriate.
- The default for *GetMaxCationPiAngle* was reduced to from 50 to 40 degrees.
- The default for *SetMaxNonIdealAcceptorAngle* was increased from 70 to 90 degrees.
- The default for *GetMaxNonIdealDonorAngle* was increased from 50 to 70 degrees.
- Several intra molecular interaction types are now properly perceived with *OEPerceiveInteractionHints*

5.44 OESystem TK 3.1.1

5.44.1 Major bug fixes

- An issue that caused the string format method on the OEFieldType for `double` to be converted to `int` has been fixed. The format method now writes with a precision of 12.
- An issue that caused the string format method on the OEFieldType for `double` to add a .0 after a `NaN` has been fixed.
- An issue that caused the string format method on the OEFieldType for `string` and `JSON` to be elided if longer than 100 characters has been fixed.
- An issue with metadata flags not getting properly updated when a field with new metadata was set on an existing field has been fixed.

5.45 OEPlatform TK 3.1.1

5.45.1 New features

- ZSTD has been updated to version 1.4.9, resulting in improvements in compression and decompression performance for the OEZ file format.

5.46 OEGrid TK 3.1.1

5.46.1 Major bug fixes

- *OEScalarGrid* now preserves *OEGridType* through I/O.

5.47 OEMath TK 3.1.1

Minor internal improvements have been made.

5.48 Version 3.1.0.2

5.48.1 OEChem TK 3.1.0.2

March 2021

- Additional metadata constants were added to the OEMetadata namespace for controlling the visibility of fields in the Orion user interface.

5.48.2 OEBio TK 3.1.0.2

- The *OEReceptorCustomConstraints.GetConstraintFeatures* method is now properly wrapped in python.
- The *GetCustomConstraints* now returns a const or non const ref instead of a copy
- The *OEReceptor.GetProteinConstraints* can now also return a non-const iterator

5.49 Version 3.1.0

5.49.1 OEChem TK 3.1.0

Fall 2020

New features

- A new function, *OEGetHydrogenCounts*, has been added that provides implicit and explicit hydrogen counts in a molecule.
- A new function, *OEApplyStateFromRef*, has been added that provides the ability to apply a certain state to a molecule. This could be used to apply tautomers to a molecule in 3D or to fix broken valence states using a known state.
- A new function, *OERGroupsToHydrogens*, has been added that converts R-groups of a molecule to hydrogen atoms.
- A new preliminary namespace, *OEFileStringType*, has been defined that allows controlling file types in *OESimpleAppOptions* and *OERefInputAppOptions*.
- Two new preliminary classes, *OEFileStringParameter* and *OEOutFileStringParameter*, have been added to allow better control over file types that can be used in *OESimpleAppOptions* and *OERefInputAppOptions*.
- The constructor of the preliminary class *OESimpleAppOptions* has been modified to take constants defined in *OEFileStringType* to better control the desired input/output file types.
- A new preliminary class, *OERefInputAppOptions*, has been added to enable creating an interface to applications and scripts that require a second input, such as a reference or target system.
- A new method, *OEResidueCategoryData.AllInDB*, has been added that determines if all the specified residue names found in the list are associated with the specified category.

Minor bug fixes

- An issue that occurred when a change in the aromaticity model perception would result in double bonds that should have lost their cis/trans parity assignments has been fixed. The bond stereo parities had been incorrectly being retained on aromatic atom endpoints.
- An issue with parsing SMILES with ExtBond couplings that were not honoring the convention that a specified bond order takes precedence over an implicit one, similar to ring closures, has been fixed.
- Extra processing has been added to ensure that a common header corruption for SDF format files throws a warning to indicate a format problem. In general, attempts to skip individual corrupted records in a multi-record file format are employed to ensure that as much of the valid input as possible can be processed. Depending on the type of corruption as well as the specific file format, there is no guarantee that a warning will be emitted for all corrupted records.
- An issue that caused *OECopyPDBData* and *OECopySDData* to avoid the copy activity if the source and destination molecules are the same molecule instance, which can cause a crash, has been fixed. The status return will be false, just like a general failure for the copy activity.
- An issue in *OEPDBOrderAtoms* that caused bad behavior when atoms in a molecule do not have any residue information has been fixed.

5.49.2 OEBio TK 3.1.0

Fall 2020

New features

- Two new methods, *OEDesignUnit.HasTransform* and *OEDesignUnit.GetTransform*, have been added that check whether a valid OETrans object is stored on the OEDesignUnit and retrieve it for use with *OETransformDesignUnit*.
- A new method, *OEMolComplexCategorizer.GetResidueCategoryData*, has been added that returns the non-const reference to the *OEResidueCategoryData* stored in the *OEMolComplexCategorizer* class.
- A new class, *OEReceptor*, has been added that represents a receptor as used in **OEDocking TK**.
- A new class, *OEReceptorConstraintFeature*, has been added that represents a constraint feature for an *OEReceptor*.
- A new class, *OEReceptorProteinConstraint*, has been added that represents a protein constraint for an *OEReceptor*.
- A new preliminary namespace, *OEProteinConstraintType*, has been added that defines the valid types for *OEReceptorProteinConstraint*.
- A new class, *OEReceptorCustomConstraints*, has been added that represents custom constraints for an *OEReceptor*.
- Two new methods, *OEDesignUnit.HasReceptor* and *OEDesignUnit.GetReceptor*, have been added that check whether a valid *OEReceptor* object is stored on the OEDesignUnit and retrieve it.
- Several new constants have been added to the namespace *OEDesignUnitComponents*, related to downstream usage, such as OEMakeReceptor.

Major bug fixes

- A bug that caused OEDesignUnits with the Iridium Category *OEIridiumCategory.NA* to be assigned *OEIridiumCategory.NT* when read from a file has been fixed.

Minor bug fixes

- A bug in *OEPlaceHydrogens* that resulted in histidine residues being in an unoptimized state, depending on the input state, has been fixed.

5.49.3 OESystem TK 3.1.0

Fall 2020

New features

- A new method, *OEOptions.AddOption*, has been added that allows adding a sub-option to an existing *OEOptions*.

Minor bug fixes

- Behavior of the field metadata on *OERecord* has been fixed to ensure that at most one field on a record can have the *ID* flag.
- An issue that caused corruption when reusing an *OERecord* in a loop has been fixed.

5.49.4 OEPlatform TK 3.1.0

Fall 2020

New features

- *OEOnce* has been deprecated. See the **OEPlatform TK** documentation for alternatives.

5.49.5 OEGrid TK 3.1.0

Fall 2020

Major bug fixes

- A bug in the symmetry operators for the I121 space group has been fixed.

Minor bug fixes

- An issue that caused the function *OEReadGrid* to incorrectly change the *OEGridType* of grids to the *OEGridFileType*, where it was being used to read `OESystem::OEFizzGrid` from a file, has been fixed. The *OEGridType* now remains the same through a round trip grid write and read.
- An issue that caused the function *OEReadMTZ* to crash when it read large MTZ files and converted them to an *OESkewGrid* as well as computing the AMIN and AMAX statistics for the resulting *OESkewGrid* incorrectly has been fixed. The code has been corrected so that the crash no longer happens for large grids; the AMIN and AMAX calculations have also been corrected.
- An issue that caused the script make_design_units.py to crash when reading PDB files using the function *OEReadMolecule*, when space group numbers that could not be recognized has been fixed. These space group numbers had been preventing the code from reading symmetry correctly from the PDB file. The code has been changed so that incorrect space group numbers result in a warning instead of a fatal crash.

5.49.6 OEMath TK 3.1.0

Fall 2020

New features

- Two new , *OEBox* and *OESphere*, have been added that represent a box and a sphere, respectively.
- The following functions have been added as access functions to the new *OEBox* class:
  - *OEBoxArea*
  - *OEBoxExtend*
  - *OEBoxTranslate*
  - *OEBoxVolume*
  - *OEBoxXDim*
  - *OEBoxXMid*
  - *OEBoxYDim*
  - *OEBoxYMid*
  - *OEBoxZDim*
  - *OEBoxZMid*
  - *OEInBox*
  - *OESetupBox*
- The following functions have been added as access functions to the new *OESphere* class:

- *OEInSphere*
- *OESphereArea*
- *OESphereVolume*

5.50 Version 3.0.0.6

5.50.1 OEChem TK 3.0.0.6

- A memory leak while reading string generic data in Python has been fixed.

5.50.2 OESystem TK 3.0.0.6

- When reading and writing large *OERecord*, sometimes memory usage would spike briefly due to temporary copies of data. The memory usage has been optimized to reduce these spikes.

5.51 Version 3.0.0.5

5.51.1 OEChem TK 3.0.0.5

- Added the ability to perceive chirality without including nitrogen atoms that could undergo nitrogen inversion. See *OEPerceiveChiral* for details.
- Added a method to clear chiral perception before calling the perception function with a different flag. See *OEClearChiralPerception* for details.
- The performance of converting large proteins to the JSON format has been improved.

5.51.2 OESystem TK 3.0.0.5

- Fixed an issue which allowed OERange objects to be created with a low value greater than the high value.
- The `readonly` flag on molecule fields was not being enforced properly in the Python API. Attempting to use a read-only field to set a molecule on an OERecord now raises an exception.
- Uninitialized OERange values stored on records were turning into None values on retrieval via the Python API. This has been fixed.
- In the Python API, the OERecord.get_fields method would return None values for fields whose types had not been imported. This has been fixed.
- Values of NaN and Infinity were causing errors when they appeared in JSON objects being read by any toolkit. This has been fixed.

5.51.3 OEGrid TK 3.0.0.5

- An issue causing some MTZ files to be read with the wrong space group has been fixed.

5.52 Version 3.0.0

5.52.1 OEChem TK 3.0.0

New features

- A new set of Preliminary API classes has been added as part of the extended set of *OEParameter* and *OEOptions* classes:

  - *OEMolParameter*
  - *OEMolReaderParameter*
  - *OEMolWriterParameter*
  - *OEPredicateParameter*
  - *OESimpleAppOptions*
- A new functor, *OEIsTermHeavyAtom*, has been added that identifies terminal heavy atoms.
- A new function, `OEOverlayConfsByRMSD`, has been added that overlays all conformers of an *OEMCMolBase* molecule on the first conformer to minimize RMSD.
- The molecule database implementation of *OESubSearchDatabase* now loads scenes into memory using multiple threads.

Minor bug fixes

- The full molecule-based multi-conformer `OERMSD` calculation now returns `false` when the automorphism flag is set to `true` and no match can be identified between the reference target molecule (i.e., when no RMSD value can be calculated).
- An issue with using deleted atom pointers as templates for creating or recreating new atoms based on the deleted atom template has been fixed. Previously, the newly created atom instance would be marked as a deleted atom.
- An issue that caused molecule SMILES string generation to erroneously output reaction information if `OERxnRole` information was present on atoms has been fixed. Reaction information should only be added to the SMILES output if `OEMolBase.IsRxn` is `true`.
- `OERxnRole.Agent` and `OERxnRole.Catalyst` enums have long had the same value, making them indistinguishable if applied to reaction atoms. The enum values have been changed to make them unique. Formats (e.g., OEB) that support importing and exporting this information can now distinguish and retain this information.
- Some of the extraneous warnings about planar stereocenter geometry have been reduced via more rigorous checking of the stereocenter environment during structure file reading.

5.52.2 OEBio TK 3.0.0

New features

- The following new methods have been added to the *OEInteractionHint* class to access the calculated geometries of the interactions by the `OEPerceiveInteractionHints` function:
  - `OEInteractionHint.HasCalculatedGeometry`
  - `OEInteractionHint.GetCalculatedGeometry`
  - `OEInteractionHint.GetCalculatedGeometries`
- A new function, `OETransformDesignUnit`, has been added to transform *OEDesignUnit* objects.

5.52.3 OESystem TK 3.0.0

New features

- A set of preliminary APIs for **OERecord** and its associated classes has been added to **OEChem TK**. **OERecord** is a data container for storing and transmitting strongly typed data and its associated metadata. The API set includes the following classes and functions:
  - *OERecord*
  - OEField

  - OEFieldBase
  - OEFieldMeta
  - OERange
  - OEWriteRecord
  - *OEReadRecord*
  - OERecordToBytes
  - OERecordFromBytes

- A new set of preliminary APIs has been added to combine the concepts of *OEParameter* with *OEOptions*. These new APIs provide an interface for configuring *OEOptions* directly from the command line, making it easier to write command line applications. The new API set includes the following:
  - *OEBoolParameter*
  - *OEConstParameter*
  - *OEDoubleParameter*
  - *OEFloatParameter*
  - *OEIntParameter*
  - *OEMultiParameter*
  - *OEPODParameter*
  - *OEPointerParameter*
  - *OEStringParameter*
  - *OEUIntParameter*
  - *OEOptions*
  - `OEConfigureItf`
  - `OEConfigureOpts`
  - `OEUpdateOpts`
- A new overload for `OEDots.Update` has been added that takes a character as an argument to display in lieu of ..

5.52.4 OEPlatform TK 3.0.0

- Minor internal improvements have been made.

5.52.5 OEGrid TK 3.0.0

- Minor internal improvements have been made.

5.53 Version 2.3.0

5.53.1 OEChem TK 2.3.0

New features (Preliminary)

- The following preliminary APIs have been added to perform fast substructure searches:

  - A new class, *OESubSearchScreen*, has been added that stores bit-vector screens. In addition, a new `OESubSearchScreenType` namespace and *OESubSearchScreenTypeBase* class have been added that handle various screen types.
  - Two new low-level functions, `OEMakeSubSearchQueryScreen` and `OEMakeSubSearchTargetScreen`, have been added that generate query and target molecule screens, respectively. The `OEScreen` low-level function can be used to determine whether a query molecule can potentially be matched to a target molecule by comparing their screens.
  - A new function, `OEIsComplementaryScreenType`, has been added that determines which screen types are suited for a specific query molecule.
  - New helper functions, `OESameSubSearchScreenTypes`, `OEGetSubSearchScreenType`, and `OEIsValidSubSearchScreenString`, have been added.
  - A new *OECreateSubSearchDatabaseOptions* option class and `OECreateSubSearchDatabaseFile` function have been added that pre-generate databases for substructure search. A new `OEIsValidSubSearchDatabase` function also has been added that checks whether a molecule file is valid for substructure search.
  - The *OESubSearchDatabase* class is designed to perform rapid substructure searches. A new `OESubSearchDatabaseType` namespace has been added to this class that enables various database implementations.
  - A new class, *OESubSearchQuery*, has been added that sets up search queries. In addition, a new *OESubSearchResult* class has been added that reports the progress of the search and retrieves the matches.
  - A new `OESubSearchStatus` namespace and `OESubSearchStatusToName` helper function have been added that monitor the search progress.

- A new low-level function, `OEReadCIFFile`, has been added that reads both small and macromolecule Crystallographic Information File (CIF and mmCIF) formats.

New features

- Two new classes, *OEConfRMSDOptions* and *OEConfRMSD*, have been added that set up *OEConfRMSD* calculations and provide fast RMSD calculation of conformers of the same molecule, respectively.
- A new function, `OEFastaToMol`, has been added that converts a FASTA sequence into a molecule.
- A new optional sorting argument has been added to the `OEGetNearestNbrs` function that allows returned atom pairs to be sorted based on distance.
- *OEIsAlphaCarbon* has been deprecated. The *OEIsCAlpha* predicate should be used instead.
- New `OEUncolorMol` overload functions have been added to *OEMCMolBase*. A new constant, `OEUncolorStrategy.RemoveDimension`, has been added that invalidates 3D coordinates and sets the dimension to `0` when uncoloring molecules.
- Support has been added to `OEReadMolecule` that enables reading both CIF and mmCIF files.
- Updates have been made to the *the OEFormat namespace* to include new CIF and mmCIF formats. Corresponding updates have also been added to `OEIFlavor` and `OEOFlavor`.
- Updates have been made to the `OEResidueIndex` namespace to include residue IDs for metals observed in PDB files.
- The formal charge assignment for less common metals observed in PDB files has been expanded.
- Updates have been made to the `OEExprType` namespace to allow for more fuzzy searches. For example, `OEExprType.ElemMinHCount` or `OEExprType.ElemMinDegree` can now be used to specify an atom's minimum hydrogen count or minimum degree.

- Changes have been made to the *OEResidue* class to support the mmCIF format. These changes include the following three new string member variables, with the appropriate getter and setter functions:
  - `SequenceID` captures the mmCIF field, *label_seq_id*, which is the sequential number of residues from 1.
  - `ExtChainID` captures the fact that the previous chainID in the *auth_asym_id* field can now have multiple characters. The *OEResidue.GetChainID* function now returns the first character of the `ExtChainID` variable, and *OEResidue.SetChainID* now overwrites and sets the `ExtChainID`.
  - `SubChainID` captures the *label_asym_id* field, which commonly seems to be a further subdividing of the *auth_asym_id* field.

Minor bug fixes

- A bug that caused *OEWriteMolReturnCode.Success* to be returned even though an invalid output stream was used has been fixed.
- A bug that caused broken perception after implicit hydrogen assignment in the *OEAssignImplicitHydrogens* and *OEAssignMDLHydrogens* functions has been fixed.
- Handling of partially resonant nitrogen stereo perception in *OE3DToAtomStereo* and *OE3DToInternalStereo* has been improved.
- An issue that caused *OEIsCTerminalAtom* to misperceive a minor number of C-terminal atoms has been fixed.
- Unpaired interactions are no longer triggered on ligands due to nearby water molecules. Similarly, water molecules can no longer present as having unpaired interactions.

Python-specific changes

- Support for making custom MCS scoring functions from *OEMCSFunc* has been added.
- New functions, `OEGetBytesFromGrid`, `OEGetScalarGridFromBytes`, and `OEGetSkewGridFromBytes`, have been added to support the conversion between byte[] and grids (*OESkewGrid* or *OEScalarGrid*).
- *OETrans.Extract* is now available in Python.

Java-specific changes

- New functions, `OEGetByteArrayFromScalarGrid`, `OEGetByteArrayFromSkewGrid`, `OEGetScalarGridFromByteArray`, and `OEGetSkewGridFromByteArray`, have been added to support the conversion between byte[] and grids (*OESkewGrid* or *OEScalarGrid*).

Documentation changes

- Two new examples have been added that demonstrate performing fast substructure searches:
  - *Generate database for fast substructure search*
  - *Perform SMARTS search on substructure database file*
- A new example program, *Printing protein amino acids histograms*, has been added that demonstrates printing a histogram of the occurrences of various amino acids in a protein.
- The *Print OEChem and file format information* example program has been updated to demonstrate using the *OEToolkitsGetRelease* function.
- New code snippets have been added to the *OEConfRMSD* class that demonstrate finding the two conformers of a molecule with the smallest RMSD.
- New code snippets have been added to the *OEFastaToMol* function that demonstrate creating a molecule from a FASTA sequence.

- New code snippets have been added to the section, *String Input and Output*, that demonstrate reading and writing molecules from memory buffers.
- New code snippets (in C++ and Python) have been added to the *OEMCSFunc* class that demonstrate using custom scoring functions.
- **OEChem TK** examples have been cleaned up to use the new C++11 style.

5.53.2 OEBio TK 2.3.0

New features

- The following APIs have been added for perceiving non-ideal hydrogen bond interactions (i.e., hydrogen bonds with long donor-acceptor distances or poor geometries):
  - A new functor, *OEIsNonIdealHBondInteractionHint*, has been added that easily identifies non-ideal hydrogen bonds.
  - New interaction types, `OEHBondInteractionHintType.NonIdealLigandAccepts` and `OEHBondInteractionHintType.NonIdealLigandDonates`, have been added.
  - The following methods have been added that customize the non-ideal hydrogen bond perceptions:
    - `OEPerceiveInteractionOptions.SetMaxNonIdealDonorAngle` and `OEPerceiveInteractionOptions.GetMaxNonIdealDonorAngle`
    - `OEPerceiveInteractionOptions.SetMaxNonIdealAcceptorAngle` and `OEPerceiveInteractionOptions.GetMaxNonIdealAcceptorAngle`
    - `OEPerceiveInteractionOptions.SetMaxNonIdealHBondDistance` and `OEPerceiveInteractionOptions.GetMaxNonIdealHBondDistance`
- A new functor, *OEIsHetAtom*, has been added that identifies atoms with the *hetero atom* property in an `OEResidues` object.
- A new functor, *OEIsDisulfideBond*, has been added that identifies disulfide bonds.
- A new function, `OECalculateDPI`, has been added that calculates the diffraction-component precision index (DPI) value.
- New categories, `OEResidueDatabaseCategory.Lipid` and `OEMolComplexFilterCategory.Lipid`, have been introduced that enable splitting lipids from a system.
- A new method, `OEInteractionHintContainer.DeleteInteractions`, has been added that removes all interactions.
- The functional group perception in `OEPlaceHydrogens` has been improved.

Minor bug fixes

- Perception of hydrogen bonds involving sulfur and anions in `OEPerceiveInteractionHints` has been improved.
- The precedence of *OEInteractionHint* perception has been improved. Chelation interactions now take precedence over charge-charge interactions.
- A check has been added to the `OEPlaceHydrogens` function that causes it to exit early if two heavy atoms are within 0.5A of each other, since this is considered to be bad input and causes unspecified behavior.

5.53.3 OESystem TK 2.3.0

New features

- The performance of *OEGetDataType*, especially on Windows, has been improved. *OEGetDataType* is now at least 500X faster when used with a single thread and about 2X faster with many simultaneous threads on Windows.
- A new method, *OEBitVector.CountRangeBits*, has been added that identifies the number of bits that are set to 1 in the given bit range.
- A new class, *OEConsoleProgressTracer*, has been added that tracks the current progress of a task.
- A new function, *OEStringEquals*, has been added that determines whether two strings are equivalent using a case-insensitive comparison.
- A new function, *OEGeom3DReflectCoord*, has been added that reflects coordinates around a plane for a hydrogen atom.
- A new function, *OEBitIsPowerOf2*, has been added that determines if a bitmask is a power of 2.

5.53.4 OEPlatform TK 2.3.0

New features

- A new *OEToolkitsGetRelease* function has been added that displays the release name of the overall toolkit version being used.

5.53.5 OEGrid TK 1.6.4

- Minor internal improvements have been made.

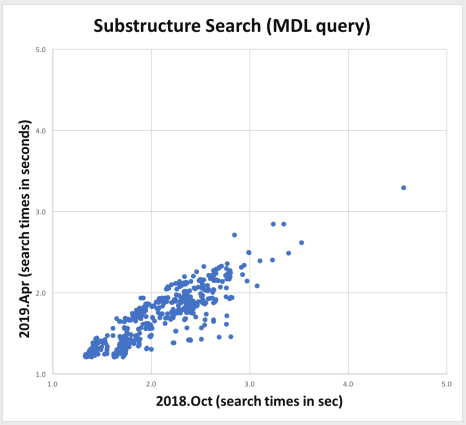
5.54 Version 2.2.0

5.54.1 OEChem TK 2.2.0

New features

- Atom and bond expressions of query molecules have been optimized for substructure search. The graphs below show the performance improvement since the 2018.Oct release. Optimization has been turned on for both MDL queries and SMARTS patterns and can be controlled by the following options:
  - *OESmartsParseOpts.Optimize* when using the *OEParseSmarts* function
  - *OEMDLQueryOpts.Optimize* when using the *OEReadMDLQueryFile* function

Table 1: **Substructure search performance improvement for MDL queries**



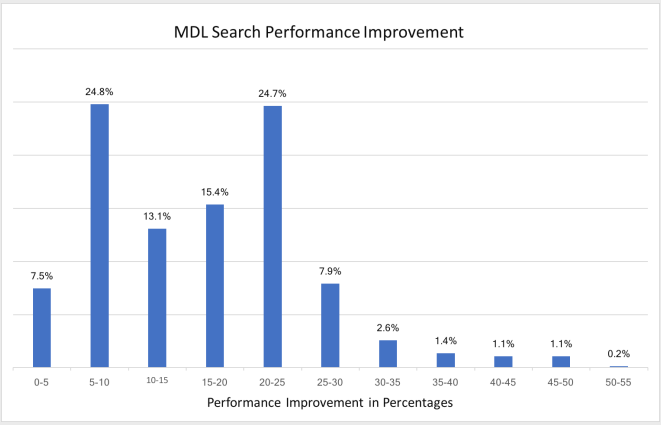
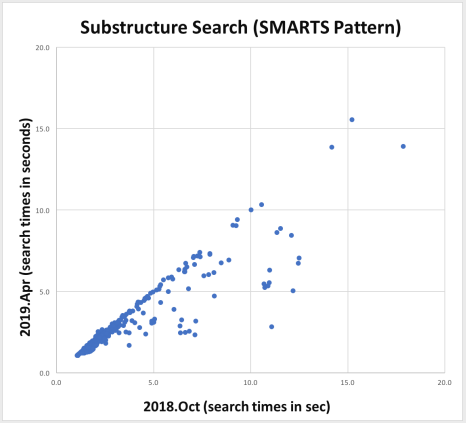
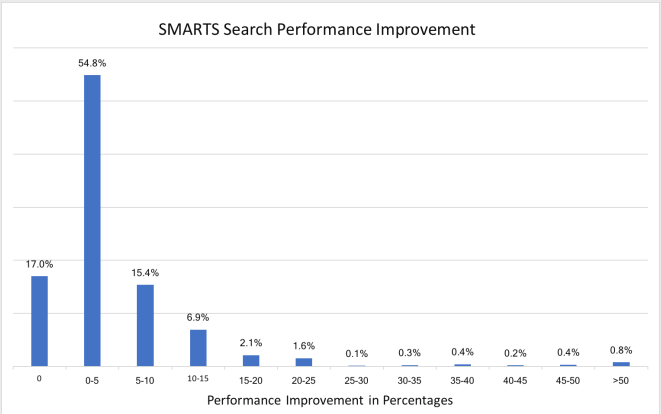


Table 2: **Substructure search performance improvement for SMARTS patterns**





- A new *OEFormat.OEZ* compressed binary format has been added with the `.oez` extension. This file format only supports multi-conformer molecules derived from the *OEMCMolBase* class. Each molecule is individually compressed in the file using the Zstandard compression algorithm. See the *Compressed Input and Output* section for more details.

> **Note**
>
> This new file format is not backwards compatible. **OpenEye** applications and toolkits released before April 2019 will *not* be able to read or write this file format.

- An *OEBitVector* object now can be attached as generic data to any *OEBase* object. Its derived classes now round-trip through the OEB format.
- Multi-conformer molecule overloads have been added to the following existing APIs:
  - *OEMDLHasParity*
  - *OEMDLGetParity*

  - *OEMDLSetParity*
  - *OEMDLClearParity*

- New input flavors, *OEIFlavor.MDL.SuppressEmptyMolSkip*, *OEIFlavor.SDF.SuppressEmptyMolSkip*, and *OEIFlavor.RDF.SuppressEmptyMolSkip*, have been added to the MDL formats to allow returning empty molecule records. This allows the SD data to be recovered from these records.
- New output flavors, *OEOFlavor.MDL.SuppressTimestamps* and *OEOFlavor.SDF.SuppressTimestamps*, have been added to the MDL formats to explicitly suppress timestamps in the file header. This is convenient for generating diffable test files.
- A new *OEResidueToString* function has been added to convert an *OEResidue* object into string representation.
- The following new classes have been added to provide alternative ways to access tagged property data in molecules:
  - *OEMolPropertyFxn*
  - *OEMolSDPropertyFxn*
  - *OEMolGDPropertyFxn*
  - *OEMolTaggedPropertyFxn*
  - *OEMolPropertyList*
- New predicates, *OEIsValidAtomValence* and *OEIsValidMDLAtomValence*, have been added to provide the ability to loop over atoms that satisfy the **OpenEye** and MDL valence model requirements, respectively.
- The *OEMolBase.NewGroup* method now also takes an *OEAtomBondSet* object.
- New *OECheckAtomValence* and *OECheckAtomValences* functions have been added that check if an atom or all atoms of a molecule have valid valences.
- A new *OELibraryGen.NumPossibleProducts* method has been added that returns the number of possible products that can be generated.
- New *OEClearSDData* and *OEClearPDBData* function overloads have been added to clear SD or PDB data of multi-conformer molecules.
- A new *OEShortestPath* overload function has been added that disallows traversing excluded bonds in the path.
- A new *OEAtomBondSet.GetParent* method has been added.
- A new *OEGetResidueIndexFromCode* function has been added.
- The *OEPDBOrderAtoms* function has been updated to take an argument that preserves the PDB residue order. It defaults to `false` to preserve backwards compatibility.

Major bug fixes

- The number of cases where round-tripping to MDL file format results in stereo information loss has been significantly reduced.

Minor bug fixes

- High-level SMILES readers such as *OESmilesToMol* and *OEReadMolecule* now clear cis/trans bond specifications on bonds in small rings (less than 7 members).
- The *oemolistream.open* and *oemolostream.open* methods now correctly set the formats for `gz` filenames.
- A bug that caused a crash when an invalid *oemolostream* is finalized has been fixed.
- *oemolistream.Setgz* and *oemolostream.Setgz* methods are now a no-op if the state of the stream is unchanged.

- An issue that caused the JSON writer to crash with user-specified cis/trans stereo has been fixed.
- The MDL V3000 reader has been fixed and no longer marks "either" double bonds with cis/trans parity.
- The MDL V2000 reader has been fixed and no longer adds "either" double marks to chiral but unspecified cis/trans double bonds.
- *OESubsetMol* no longer crashes when breaking and adding multiple R-groups to the same atoms.
- Cobalt ions are now assigned a formal charge when being read from the PDB file.

Python-specific changes

- The `__str__` method of the *OEResidue* class now invokes the new *OEResidueToString* function.
- The *OEMolDatabase.GetMoleculeString* method has been fixed and now returns `bytes`.

Documentation changes

- New code snippets have been added to the *Substructure Search* section to show how to reconstruct matches for the original molecule.
- A new *Split molecule file* example has been added.
- New code snippets have been added to the *OEChemIsLicensed* function.

5.54.2 OEBio TK 2.2.0

New features

- A new *OEIsCTerminalAtom* predicate has been added to identify the C-terminal atom(s) in a protein structure.
- A new *OEIsNTerminalAtom* predicate has been added to identify the N-terminal atom(s) in a protein structure.
- A new *OEAtomMatchResidue* predicate has been added to identify residue atoms by RegEx.
- New *OEGetResidueAtom* functions have been added to provide easy access for specific residue atoms.
- The *OEIsBackboneAtom* predicate has been updated to take an optional argument to consider the terminal oxygen atom (OXT) as part of the backbone.
- The *OEIsWater* predicate has been updated to take two optional arguments:
  - to match hydrogen atoms in water molecules
  - to match the water ions OH^- and H_3O^+
- A new *OEIsStandardProteinResidue* overload that takes an atom has been added.

New features (Preliminary)

- The following preliminary APIs have been added to handle design units:
  - *OEDesignUnit*
  - *OEDesignUnitFileType*
  - *OEIsReadableDesignUnit*
  - *OEIsWriteableDesignUnit*
  - *OEReadDesignUnit*
  - *OEWriteDesignUnit*
  - *OEDesignUnitComponents*

- *OESubsetDesignUnit*
- *OEUpdateDesignUnit*
- *OEIridiumCategory*
- *OEIridiumData*
- *OEGetIridiumCategoryName*
- *OEGetIridiumCategoryFromName*
- *OEStructureQuality*
- *OEGetDesignUnitComponentID*
- *OEGetDesignUnitComponentName*

The above classes and functions are currently used by **Spruce TK** for protein handling.

Minor bug fixes

- The *OEPlaceHydrogens* function has been improved:
  - The *OEPlaceHydrogensMoverClass.NH2* mover perception has been updated.
  - The *OEPlaceHydrogensMoverClass.COOH* mover perception has been updated to only mark the atoms involved in the mover.

Java-specific changes

- The Java example in the *Splitting a Macro-molecular Complex Efficiently and Flexibly* section has been fixed.

Documentation changes

- A new *Interaction Perception* chapter has been added.
- Code snippets have been added to the *OESetCrystalSymmetry* and *OEGetCrystalSymmetry* functions.

5.54.3 OESystem TK 2.2.0

Minor bug fixes

- The *OEMakeMolecularGaussianGrid* function now returns `false` for an empty molecule.
- New code snippets have been added to the *OEThrow* class to demonstrate how to redirect warning and error messages to a file.

5.54.4 OEPlatform TK 2.2.0

New features

- The following APIs have been rewritten using standard C++11 classes:
  - *OEMutex* now uses std::mutex.
  - *OETryMutex* now uses std::try_lock.
  - *OELock* and *OELockSpinlock* now use std::lock_guard.

Minor bug fixes

- The *OEAddLicenseFromHttp* function available in the wrapped languages (Python, Java, and C#) now accepts HTTP responses with a `text/plain; charset=utf-8` content type.

5.54.5 OEGrid TK 1.6.3

Minor bug fixes

- The *OEReadMTZ* function has been updated to read CCP4 alternative space group numbers (e.g., 4005).
- *OEReadGrid* no longer loses its title beyond the first blank space.
- A potential memory corruption issue with Gaussian volume calculation has been fixed.

5.55 Version 2.1.5

5.55.1 OEChem TK 2.1.5

New features

- New *OEHasExplicitHydrogenExpression* and *OEHasImplicitHydrogenExpression* functions have been added.
- The *OEPrepareSearch* function now takes a parameter that determines whether to adjust the hydrogen of a target molecule based on the presence or absence of implicit and explicit hydrogens on the query molecule of the substructure search.
- Both *OEGetSubtree* and *OEGetSmallestSubtree* functions now take an optional max depth parameter that allows partial traversal of subtrees. For example, the parameter allows identification of the subtree of a proline residue without traversing back onto the main chain. Similarly, it is possible to get the subtree of a cysteine residue involved in a disulfide without traversing across the disulfide bond and back on the main chain.
- Two new constants, *OEIFlavor.MDL.SuppressImp2ExpENHSTE* and *OEOFlavor.MDL.SuppressImp2ExpENHSTE*, have been added with related MDL input and output formats (e.g., SDF, RDF) to allow suppression of explicit MDL stereogroup generation for implied stereocollection membership. By default, marked stereocenters not explicitly included in stereocollections are added to their implied stereocollection membership based on the chiral flag setting.
- A new *OEGroupType.UserDefined* type has been added to *OEGroupBase* to support serialization and deserialization via the `OEB` format.
- A new *OEIsBridgeHead* functor has been added to identify bridge head atoms in a molecule.
- A new *OERingBondCount* function has been added that returns the number of bonds connected to an atom that are part of a ring.

Major bug fixes

- The FASTA reader has been fixed to support reading multiline FASTA strings.
- Generic data attached to *OEGroupBase* is no longer lost when the molecule is copied.
- Generic data attached to *OEGroupBase* is no longer lost when the molecule is added to another molecule by invoking the *OEAddMols* function.
- Relative stereo (e.g., 1,4-disubstituted cyclohexanes) information is now preserved on input, ensuring that the conformers generated will respect the input stereochemistry. Previously, such relative stereocenters would be cleared during activities to clear other illegally marked stereochemistry.

- The V3000 MDL format readers called by *OEReadMolecule* now support noncontiguous object indices. It previously had been assumed that such indices were contiguous, but recent third-party writers have demonstrated that this assumption is no longer valid.

Minor bug fixes

- The *OEGenerate2DCoordinates* function now returns `false` when it fails to generate coordinates that are consistent with the atom/bond stereo of a molecule.
- A memory corruption issue in the InChI generation code that previously required a check for hydrogen isotopes beyond `[3H]` will no longer generate InChI for these unrealistic hydrogen isotopes.
- Malformed SD bond lines that lacked both atom endpoints had previously been ignored. *OEReadMolecule* now fails for these types of format errors.
- *OEUniMolecularRxn* internal validations have been relaxed to allow reaction transformations that create a bond between two separate fragments.
- *OEGetHybridization* has been repaired to return *OEHybridization.sp3d* and *OEHybridization.sp3d2* for high valence halogens. Previously, this function had only returned *OEHybridization.sp3* for these cases.
- *OEUncolorMol* has been augmented to remove stereogroups when stripping atom stereochemistry. Additionally, a new option, *OEUncolorStrategy.RemoveGroupStereo*, has been added to support the explicit stripping of stereogroups.
- *OEAnd* and *OEOr* have been augmented to support using *OEGroupBase* types to allow a more complex selection of MDL stereogroup types via custom predicates.
- A new copy constructor and new assignment operator have been added to *OERotMatrix*.

Python-specific changes

- Type comparisons between `OEBase` and non-`OEBase` objects have been fixed to ensure that an exception is longer thrown.
- A new `__str__()` function has been added to *OEResidue* to provide a convenient string representation.

Java-specific changes

- A new `OELibExtractor.GetTmpDirName()` method has been added to return the temporary directory used for the extraction of native libraries from the JarFile and the extraction lock file. This directory location can potentially be used to clean up extraction locks that have been left behind due to inadvertent or explicit Java process kill activities.

Documentation changes

- The documentation for *OESMILESFlag.Hydrogens* has been clarified.
- The porphyrin footnote in the aromaticity table in the *Aromaticity Perception* section has been clarified.

5.55.2 OEBio TK 2.1.5

New features

- The following APIs have been added to perform Ramachandran analyses on protein structures:
  - *OERamachandranAnalysis* class
  - *OEGetRamachandranTypeName* and *OEGetRamachandranCategoryName* functions
  - *OERamaType* and *OERamaCategory* namespaces

- A new *OERotamerLibrary.Richardson_2016* constant has been added to provide rotamers from the ultimate rotamer library [Hintze-2016] . This new rotamer library is now the default.
- The following atom predicates have been added to identify RNA atoms:
  - *OEIsNucleicAcidBase*
  - *OEIsNucleicAcidSugar*
  - *OEIsNucleicAcidPhosphate*
- A new *OESetRotamer* overload has been added that takes a residue atom.

Minor bug fixes

- It is now possible to set an alternative rotamer on proline residues.

C++-specific changes

- **OEBio TK** now depends on **OEGrid TK**. As result, CMake files might need to be modified to reflect this library dependency.

Java-specific changes

- The Java wrapping of the following predicates has been fixed:
  - *OEIsCationPiInteractionHint*
  - *OEIsCovalentInteractionHint*
  - *OEIsIntermolecularChelatorInteractionHint*

C#-specific changes

- The C# wrapping of the following predicates has been fixed:
  - *OEIsCationPiInteractionHint*
  - *OEIsCovalentInteractionHint*
  - *OEIsIntermolecularChelatorInteractionHint*

5.55.3 OEPlatform TK 2.1.5

- Minor internal improvements have been made.

5.55.4 OESystem TK 2.1.5

- The *OEWriteSettings* function now takes a 4th argument that specifies the name of an optimized parameter set for **OpenEye**'s applications.

5.56 Version 2.1.4

5.56.1 OEChem TK 2.1.4

New features

- Two new overload functions, `OEChem::OEReadMolFromString` and `OEChem::OEWriteMolToString`, have been added to specify input or output flavors from the *OEIFlavor* and *OEOFlavor* namespaces.

- A new class, *OE2DRingDictionaryCreatorOptions*, has been added to customize generating user-defined ring template dictionaries.
- A new constructor, *OE2DRingDictionary*, has been added that takes an *OE2DRingDictionaryCreatorOptions* option class as a parameter.
- A new function, *OEClear2DRingDictionary*, has been added to clear the global user-defined ring dictionary used for 2D coordinate generation.
- A new function, *OESet2DHydrogenGeom*, has been added to assign approximate 2D coordinates to explicit hydrogen atoms.

Major bug fixes

- The `OEChem::OEReadMolFromString` and `OEChem::OEWriteMolToString` functions use internal buffers for the encoding or decoding of molecules from bytes. Previously, depending on library load order, internal buffers could be initialized without the complete set of binary input/output handlers. This could result in the loss of custom datatypes when translating to or from bytes. These internal buffers now are updated during subsequent loads of **OpenEye** libraries to ensure that the complete set of custom datatypes is handled correctly.
- The implementations of `OEChem::OEDBMol` and `OEChem::OEMiniMol` have been corrected to avoid a potential crash after an *OEMolBase.Compress* and *OEMolBase.UnCompress* call sequence when the molecule contains *OEGroupBase* objects (e.g., reaction component information or V3000 enhanced stereo information). While the `OEChem::OEMiniMol` supports *OEGroupBase* objects, it cannot save and restore following *OEMolBase.Compress* and *OEMolBase.UnCompress* calls, so this class must never be used if *OEGroupBase* information is to be maintained. The `OEChem::OEDBMol` implementation is the preferred class for low memory footprint applications when *OEGroupBase* may be present.
- When writing a 2D molecule with explicit hydrogens to an MDL file, the coordinates of the stereo hydrogens are now reassigned by calling the *OESet2DHydrogenGeom* function if they are identical to their parent heavy atom coordinates. This ensures that reinterpretation of the MDL file does not inadvertently flip stereo configurations.

Minor bug fixes

- The algorithm used by *OEGenerate2DCoordinates* to detect and resolve atom clashes when generating 2D coordinates has been improved.
- When calling the *OERMSD* function for a multi conformer fit molecule, the returned RMSD array will now have a -1.0 value for conformation indices that are not valid for the fit molecule.
- The *OEAssignAromaticFlags* function now limits the search path to 30 bonds when the bond/atom ratio is higher than 1.3. As a result, buckyball-like molecules can be processed in reasonable time.

C++-specific changes

- A deprecated overload of the *OEPRECompress* function, `OEPRECompress(OEMol &, const OEMCMolBase &)`, has been removed from the C++ API. `OEPrepareFastROCSMol` should now be used to create molecules optimized for **ROCS**-type calculations. The documentation has been updated to reflect this change.

Python-specific changes

- Two new overload functions, *OEReadMolFromBytes* and *OEWriteMolToBytes*, have been added to specify input or output flavors from the *OEIFlavor* and *OEOFlavor* namespaces.
- A crash in the Python interpreter due to malformed UTF-8 input character sequences has been fixed.
- Binary data now can be safely attached to objects as generic data without being truncated at a `\0` character.

Java-specific changes

- Two new overload functions, `OEChem::OEReadMolFromByteArray` and `OEChem::OEWriteMolToByteArray`, have been added to specify input or output flavors from the *OEIFlavor* and *OEOFlavor* namespaces.

C#-specific changes

- Two new overload functions, *OEReadMolFromBytes* and *OEWriteMolToBytes*, have been added to specify input or output flavors from the *OEIFlavor* and *OEOFlavor* namespaces.
- A minor issue with the C# `verify.py` script that caused the validation to halt if one or more toolkit licenses was missing from the active license file has been fixed.

5.56.2 OESystem TK 2.1.4

Minor bug fixes

- The *OEColor* constructor and *OEColor.SetRGBA* method now throw warnings for a null pointer.

5.56.3 OEBio TK 2.1.4

New features

- A new method, *OEInteractionHintFragment.GetAtom*, has been added.
- A new predicate, *OEIsIntermolecularChelatorInteractionHint*, has been added.

Minor bug fixes

- The *OEIsChelatorInteractionHint* predicate has been fixed to recognize the *OEChelatorInteractionHintType.IntramolecularProtein* type.
- The *OEHBondInteractionHintType.Clash* type has been deprecated and replaced with *OEHBondInteractionHintType.ClashAcceptorAcceptor* and *OEHBondInteractionHintType.ClashDonorDonor* types.
- The *OESaltBridgeInteractionHintType.UnpairedLigand* type has been deprecated and replaced with *OESaltBridgeInteractionHintType.UnpairedLigandPositive* and *OESaltBridgeInteractionHintType.UnpairedLigandNegative* types.
- The *OESaltBridgeInteractionHintType.UnpairedProtein* type has been deprecated and replaced with *OESaltBridgeInteractionHintType.UnpairedProteinPositive* and *OESaltBridgeInteractionHintType.UnpairedProteinNegative* types.
- The implementation of *OEHBondInteractionHintType.Clash* has been corrected to use proper geometric constraints in determining these unfavorable interactions. In addition, *OEHBondInteractionHintType.Clash* has been split into *OEHBondInteractionHintType.ClashDonorDonor* and *OEHBondInteractionHintType.ClashAcceptorAcceptor* to provide more specificity. These interactions are assigned only after other stronger or more favorable interactions have been assessed. Finally, hydrogen-bonding interaction clashes will not be assigned to groups that could avoid the interaction with a simple hydrogen rotation.

5.57 Version 2.1.3

5.57.1 OEChem TK 2.1.3

New features

- The InChI code has been updated to the v1.05 source released on January 27, 2017. This code is available from the InChI Trust and was validated against eMolecules and ChEMBL_23 public structure sources.
- InChI is now a fully supported read/write format for chemical structures. New public API points have been added to support this format: *OEMolToSTDInChI*, *OEMolToSTDInChIKey*, *OEInChIToMol*, and *OEParseInChI*.

> **Warning**
>
> The standard InChI format includes chemical normalization activity that results in round-trip structure alterations to a very high percentage of structures when unnormalized or when arbitrary input chemistry is used. Even the output of *OEMolToInChI*, which generates a nonstandard InChI and attempts to minimize structure normalizations, results in many changes to input chemical structures.

- *OEInChIOptions* now supports relative stereo and racemic stereo InChI output as well as control of the timeout condition.
- The default output flavor for writing InChI or InChIKey via *OEWriteMolecule* has been changed to include the molecule title as well as the InChI information to synchronize behavior with other output formats. Since the InChI identifiers never include white space, there should be no ambiguity for separating the InChI information from the title. If there are third-party readers that have problems with the title, the following output flavor bits are now available to control whether the title is added to the output: *OEOFlavor.INCHI.Title* and *OEOFlavor.INCHIKEY.Title*.
- An overload of *OEShortestPath* has been added to take an atom predicate and return the shortest path between atoms that are not excluded by the predicate.

Minor bug fixes

- *OEParseSmiles* no longer throws an "Unable to Kekulize SMILES" warning in quiet mode (see also *OEParseSmilesOptions*).
- *OEDeleteEverythingExceptTheFirstLargestComponent* has been reimplemented using *OEDetermineComponents* and no longer returns `true` for one-component molecules with explicit hydrogens.
- *OEWriteMolecule* and *OEWriteConstMolecule* now return the new *OEWriteMolReturnCode.NoConformer* constant in cases when a multi-conformer molecule is being written without any conformer.
- *OEGetSmiStringOrder* now has both a `const` and a `non-const` version.
- *OEGetSmiStringOrder* could crash for structures that were modified by deleting atoms. This has been fixed.
- The *OESMILESFlag.AllBonds* flavor flag had been missing from the *OEOFlavor* namespace and has been added to the *OEOFlavor.SMI*, *OEOFlavor.ISM*, *OEOFlavor.CAN* and *OEOFlavor.USM* namespaces.

5.57.2 OESystem TK 2.1.3

New features

- A new parameter has been added to the constructor of *OEDots* that enables changing the "processed" label in the output lines.

Minor bug fixes

- The function *OEGetTag* and method *OEBase.GetData* could segfault when called with a null `char *` argument in C++ or with an empty argument in the wrapped languages. These have been fixed.

5.57.3 OEBio TK 2.1.3

New features

- *OESetupSplitMolComplexOptions* now has two additional parameters, `siteOverride` and `modelOverride`, that make it easier to configure filters while iterating over binding sites and/or model numbers. The new constant *OESplitMolComplexParam.DoNotOverride* is used control the behavior of these constants.
- A new atom predicate, *OEIsWater*, has been added.
- Two new methods, *OEInteractionHint.GetDetails* and *OEInteractionHint.SetDetails*, have been added.
- Two new interaction hint types have been added to interaction perceptions: *OECovalentInteractionHint* and *OECationPiInteractionHint*. When covalent interactions are present, nearby clash interactions are no longer perceived.
- The *OECrystalSymmetryParams* class now stores the Z-value parsed from the `CRYST1` line of a PDB file.

Minor bug fixes

- *GetGroups* no longer crashes on Windows in cases where no alternate location group exists.
- Calling the function *OEBuildUnitCell* on protein molecules had previously consumed large amounts of memory and, as a result, would often crash. This issue has been addressed, and the performance of the function has been improved for molecules of small or medium size.
- Interaction perception for anionic carbonyl groups has been fixed.
- *OEIsValidActiveSite* now takes into account the non-default parameter ligand component type.

Documentation changes

- An example in *Preparing a Protein* has been modified to work properly when splitting a ligand from a binding site other than the first one.

5.57.4 OEGrid TK 1.6.0

Minor bug fixes

- *OEMaskGridByMolecule* no longer segfaults for an empty molecule or for a molecule without 2D coordinates. It now throws a warning and returns `false`.
- On Windows, UTF8 file and directory names are now correctly supported for toolkit file I/O operations.

Java-specific changes

- The following methods are now wrapped correctly as `constCall`:
  - *OEIsRxnComponentGroup.operator()*
  - *OEIsReactantGroup.operator()*
  - *OEIsProductGroup.operator()*

Documentation changes

- Read/write examples have been added to demonstrate the use of the InChI structure format.
- Examples of using the preferred *OEMolToSTDInChI* and *OEMolToSTDInChIKey* API points when creating standard InChI strings have been updated.

- Examples have been updated to now use *OEDeleteEverythingExceptTheFirstLargestComponent* rather than the deprecated function *OETheFunctionFormerlyKnownAsStripSalts*.
- The classes *OENearestNbrs* and *OENbrs*, function *OEGetNearestNbrs*, and namespace *OENearestNbrsMethod* have been documented.

5.58 Version 2.1.2

5.58.1 OEChem TK 2.1.2

New features

- A new function, *OEGetSmiStringOrder*, has been added to retrieve the atoms of a molecule in the order that they are encoded in the SMILES string for that molecule.
- New methods *OELibraryGen.SetClearCoordinates* and *OEUniMolecularRxn.SetClearCoordinates* have been added to clear the coordinates of generated products.

Minor bug fixes

- The function *OECount* has been fixed to return the number of substructure search matches. An extra parameter has been added to control whether the count of unique or non-unique matches is returned.
- The following functions have been fixed to properly handle multi-conformer molecules with no active conformation:
  - *OEMolBase.SetDimension*
  - *OEMolBase.GetDimension*
  - *OEMolBase.SetEnergy*
  - *OEMolBase.GetEnergy*
- Several problems with the internal implementation of *OEMolBaseType.OEMiniMol* have been fixed. Perception flags are now cleared correctly when deleting atoms or bonds. Coordinates are no longer lost when a molecule is swept.
- The *OEGenerate2DCoordinates* function now clears MDL bond stereo that can be re-perceived by calling the *OEMDLPerceiveBondStereo* function.
- The *OESubSearch.Init* method now checks that the given query molecule is valid. In case of an invalid molecule a warning is thrown and the method returns `false`.

Python-specific changes

- The message for the `ValueError` exception raised when trying to attach an unrecognized type to an **OpenEye** object as generic data has been clarified.

Documentation changes

- The documentation for the following APIs has been improved:
  - *OEAddMols* function
  - *OEWriteConstMolecule* function
  - *OEWriteMolecule* function
  - *OEParseSmiles* function
  - *OESubsetMol* function

  - *OEPDBAtomName* namespace

5.58.2 OEBio TK 2.1.2

New features

- A new method, *OEInteractionHintContainer.DeleteInteraction*, has been added.
- The set of water orientations sampled by *OEPlaceHydrogens* is now strictly based on local donor/acceptor interactions instead of the more generic quaternion sampling used previously.
- The following methods have been added to *OEPlaceHydrogensOptions* to control the optimization of conformations of moveable functional groups by *OEPlaceHydrogens*:
  - *SetBypassPredicate*
  - *SetFlipPredicate*
  - *SetNoFlipPredicate*
  - *GetBypassPredicate*
  - *GetFlipPredicate*
  - *GetNoFlipPredicate*
- The new method *OEPlaceHydrogensMoverNote.IsRestrained* indicates whether a given moveable functional group has been restrained.
- Extended chelation interactions to cases where a protein is chelating a ligand metal.
- For interaction hints between any two atoms, implement a perception precedence as follows: `Chelation` > `Charge` > `Hydrogen-bonding`.
- Added chelation perception to protein intramolecular interaction hints. See *OEChelatorInteractionHintType.IntramolecularProtein*.

Major bug fixes

- *OEPlaceHydrogens* results have been significantly improved after fixing a bug related to the treatment of certain non-moving background atoms.
- *OEPlaceHydrogens* no longer fails with a segmentation fault when encountering an H-bond where there is a missing covalent bond. It also no longer fails in certain cases when adding explicit hydrogens.
- *OEPlaceHydrogens* now uses a more concise definition of moveable groups and no longer fails due to matching atoms multiple times.
- The MMFF-NIE scoring system used by *OEPlaceHydrogens* now works with elements such as *Mn* that are not parameterized in the MMFF94 force field.
- *OEPlaceHydrogens* now generates correct hydrogen geometry on carboxylic acid (COOH) groups.
- *OEPlaceHydrogens* no longer orients phenolic hydrogens at high angles to the ring plane.
- *OEPlaceHydrogens* generates improved orientations for amines deprotonated upon chelating a metal.

Minor bug fixes

- The ordering of *moveable groups* associated with a cluster in an *OEPlaceHydrogensDetails* object is now consistent across operating systems and platforms.
- *OEPlaceHydrogens* now produces fewer warning messages.
- Improved specificity of OEAcceptor definitions for interaction hint perception.

5.59 Version 2.1.1

5.59.1 OEChem 2.1.1

New features

- A new functor class, *OESimilarByRMSD*, has been added that checks similarity between a pair of conformers based on RMSD.

Major bug fixes

- On Windows, a defect that caused a crash in *OEReadMolecule* for *OEB* format or *OEReadOEBFile* when processing large amounts of *OEFormat.SDF* or *OEFormat.PDB* data has been fixed.
- A crash that occurred when explicitly calling *oemolostream.Setgz* after opening a file that was implicitly gzipped has been fixed.
- Water hydrogens created by *OEAddExplicitHydrogens* had previously been assigned the incorrect names "H 2" and "H 3". This has been corrected to give these hydrogens the expected names "H1" and "H2". The *PDB atom indices* OEChem::OEPDBAtomName::H1 and OEChem::OEPDBAtomName::H2 can be used to refer to them. The constant OEChem::OEPDBAtomName::H has been deprecated.

Minor bug fixes

- The algorithm used by the *OEPerceiveBondOrders* function for detecting sp2 hybridized planar ring systems has been improved.
- For certain complex ring systems, the function *OECreateSmiString* previously attempted to write a SMILES with more than 99 ring closures and output an invalid SMILES. It now throws a warning and returns an empty string.
- If the function *OECreateSmiString* cannot generate a valid SMILES for a molecule, a molecule title will no longer be output.
- When extracting ring templates for depiction, the *OE2DRingDictionary* class's normalization process no longer rejects long or short bonds. This allows the creation of non-regular ring templates.
- The *OECenter* overload for *OEMCMolBase* now optionally accepts an array which, when present, returns the translations applied for each conformer during the centering operation. The additional overload does not affect any existing usage.
- The *OESymmetryNumber* overload that works with *OEMCMolBase* had previously not properly calculated the symmetry numbers for all conformers. This has been fixed.

Java-specific changes

- The Java `verify` script has been updated to correctly identify a **Szybki TK** license.
- The Java `verify` script has been updated to return an error status if there was a problem when generating a subset jar.

C#-specific changes

- The Python-based C# `verify` script has been modified to run correctly with either Python 2 or Python 3.

Documentation changes

- Documentation for *OECenter* and other molecular geometry functions has been improved.

5.59.2 OEBio 2.1.1

New features

- A new scoring system, `MMFF-NIE` ("MMFF Neighbor Interaction Energies"), has been added to improve optimizing hydrogen bonding networks and avoiding clashes in the function *OEPlaceHydrogens*. This physics-based scoring system includes Coulombic and Van der Waals terms from `MMFF94`, including MMFF partial charges. Interactions are scored between moveable functional groups (see *OEPlaceHydrogensMoverClass*) as well as between moveable functional groups and non-moving neighboring background atoms. The optimizer has also been improved to be much more memory-efficient.
- *OEPlaceHydrogens* now samples waters more densely by default. A new constant, *OEPlaceHydrogensWaterProcessing.Focused*, can be used to revert to the more limited sampling. Waters are processed individually, in order, starting at the molecular surface and moving outward, after all other moveable functional groups have been optimized.
- *OEPlaceHydrogens* now detects bad clashes more accurately.
- *OEPlaceHydrogens* now deprotonates certain acidic groups (e.g., SH) in the presence of a metal. In addition, imidazoles can now be doubly deprotonated when between two metals. A new constant, *OEPlaceHydrogensMoverClass.AroN*, has been added to describe aromatic nitrogens that can be deprotonated by metals.
- *OEPlaceHydrogens* now recognizes neutral carboxylic acids. Hydrogen positions are sampled on both oxygens.
- *OEPlaceHydrogens* now includes alternate location codes in atom descriptions.
- Two new methods, *OEPlaceHydrogensOptions.SetMaxSubstateCutoff* and *OEPlaceHydrogensOptions.GetMaxSubstateCutoff*, have been added to control how many substates the optimizer will consider in a single step. This prevents calculations from running long because of overly dense networks.
- Class *OEPlaceHydrogensOptions* objects can now be compared for equivalence and non-equivalence.

Minor bug fixes

- Previously, *OEPlaceHydrogens* was limited to processing 1,024 moveable functional groups. This limitation has been removed.
- *OEPlaceHydrogens* has been improved for adding explicit hydrogens to molecules with multiple alternate locations when *OEPlaceHydrogensOptions.SetAltsMustBeCompatible* is set to `true`, issuing a warning when it fails rather than halting.

Documentation changes

- A description of the *OEPlaceHydrogens* scoring system `MMFF-NIE` in the documentation for *OEPlaceHydrogensDetails.Describe* has been added and includes a link from the *Protein Preparation* example.
- The *Preparing a Protein* example has been modified to cover the expanded range of options in water processing as well as the maximum number of optimization substates. It also now repairs bonding mistakes when the command line option `-alt compare` is used to retain all alternates by calling the method *OEAltLocationFactory.GetSourceMol*. This is a recommended approach for other workflows that read *OEFormat.PDB* files and retain all alternate location atoms.

5.59.3 OEPlatform 2.1.1

Minor bug fixes

- The *OEFile* API has been augmented to support large (>2Gb) files on Windows.

Java-specific changes

- The SWIG Java wrapper no longer creates finalizers for **OpenEye** objects that do not own memory: `OEBase`, `OEAtomBase`, `OEBondBase`, `OEConfBase`, and `OEGroupBase`. Previously, the SWIG wrapper created empty finalizers, which resulted in garbage collection bottlenecks for processes that used a lot of objects (e.g., looping over atoms and bonds in a protein). This fix improves overall throughput for Java toolkit programs.

5.59.4 OESystem 2.1.1

Minor bug fixes

- The *OEGeom2DRotate* method previously assumed inputs with 3-dimensional coordinates. It now assumes 2-dimensional coordinates.
- The *OEGeom2DTranslate* method previously assumed inputs with 3-dimensional coordinates. It now assumes 2-dimensional coordinates.

Documentation changes

- Documentation for the public 2D and 3D geometry functions has been improved.

5.60 Version 2.1.0

5.60.1 OEChem 2.1.0

New features

- The *OEB* format has been extended to support MDL enhanced stereogroup information.
- The *OEB* format has been enhanced to support reaction atom-atom mapping and reaction component information.
- The *OEGroupBase* API has been augmented to support reaction component information. In addition, new predicates have been added to loop over the new *OEGroupBase* collection types. See *OEGroupBase.IsEmpty*, *OEGroupBase.GetGroupDefinition*, *OEHasGroupType*, *OEIsRxnComponentGroup*, *OEIsReactantGroup*, *OEIsProductGroup*.
- The *SDF* reader is now more tolerant of molfile header lines that exceed the specification maximum of 80 characters. A warning is issued on read and the longer lines are truncated to 80 characters on write to prohibit generating files that do not meet the format specification.
- New functions for computing weighted RMS deviation between pairs of Cartesian coordinates, molecules, and partial molecules have been added. See *OEWeightedRMSD* for details.

Minor bug fixes

- The tag used to identify style data in *OEB* files has been changed. The old tag was not updated when the data structure was modified last year. As a result, *OEB* files with style data that were generated before 2015.Jun cannot be read with more recent toolkits. Data with the old tag will be ignored. Since style data is only used internally, we do not anticipate any read problems with *OEB* files generated by our customers.
- Functions *OEGetFileExtension* and *OEIsGZip* now recognize capitalized extensions (for example, `.GZ`).
- Several problems with the internal implementation of *OEMolBaseType.OEMiniMol* have been fixed.

- A problem in the *SDF* reader where uninitialized memory was being accessed during the perception of V3000 reaction files has been fixed.
- A problem in the *SDF* writer relating to relative stereochemistry has been fixed. Previously, the writer incorrectly exported an input V3000 format file as a V2000 format output, resulting in the incorrect promotion of racemic stereocollection information to absolute stereochemistry.
- A problem with the *MOL2* reader that limited input lines to 80 characters has been fixed. The reader now supports 512 character lines for this format.

C++-specific changes

- New overloads of `OEChem::OEWriteMolToString` in C++ allow more efficient encoding of molecules to strings.

Python-specific changes

- Python functions `OEChem::OEWriteMolToBytes` take advantage of the new overloaded versions of the `OEChem::OEWriteMolToString` C++ functions to improve encoding performance.

Documentation changes

- Examples have been added to the *SMARTS* chapter that illustrate how to match specified and unspecified stereocenters.
- The description of *oemolistream* formats has been clarified.

5.60.2 OEBio 2.1.0

New features

- *OEPlaceHydrogens* is a new function that adds hydrogens to a molecule and builds hydrogen bonding networks, flipping amide and imidazole groups if necessary.
- The following API has been added to customize and improve the perception of potential protein-ligand interactions:
  - *OEPerceiveInteractionOptions* class
  - *OEHalogenBondInteractionHint* class
  - *OEIsHalogenBondInteractionHint* predicate
  - *OEIsIntermolecularHBondInteractionHint* predicate
  - *OEIsIntramolecularHBondInteractionHint* predicate
  - *OEHalogenBondInteractionHintType* namespace (new)
  - *OESaltBridgeInteractionHintType* namespace (extended)
  - *OEHBondInteractionHintType* namespace (extended)
- A variant of the *OESplitMolComplex* function enables a more efficient two-step process. The function *OEGetMolComplexFragments* analyzes a molecule and returns an *OEAtomBondSet* vector describing the role of each fragment. Functions *OECombineMolComplexFragments* and *OEFilterMolComplexFragments* then use this information to create new subset molecules.

Major bug fixes

- The class *OEMolComplexCategorizer* now recognizes small peptides.

Minor bug fixes

- The function *OEClearMolComplexSDData* now correctly identifies data by tag prefixes.
- The function *OEHasCrystalSymmetry* now returns `false` if the molecule has a default `CRYST1` record. PDB format rules lead to these default records, which should be interpreted to mean the molecule has no symmetry record.

Documentation changes

- The description of the output parameters for the function *OEExpandCrystalSymmetry* has been improved.
- The *Protein Preparation* chapter has been expanded and an example, *Preparing a Protein*, has been added.
- The file type in the warning associated with the functions *OESplitMolComplex* and *OEGetMolComplexComponents* has been clarified.

5.60.3 OEPlatform 2.1.0

New features

- Functions *OESetAllocaThreshold* and *OEGetAllocaThreshold* have been added to manage stack memory usage. Rather than keeping track of stack usage, which impacts performance, the new functions set/get the threshold for the largest single allocation from the stack. Above the threshold, memory is allocated from the heap. Although this only indirectly controls total stack usage, it is lightweight and can help mitigate stack memory problems.

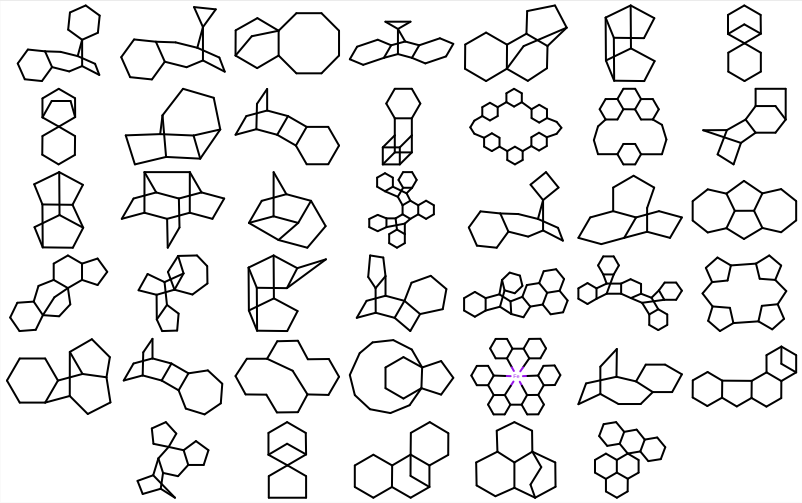
5.61 Version 2.0.7

5.61.1 OEChem 2.0.7

New features

- *OEGetPathLength* function's performance has been significantly improved.
- The APIs of the functions for accessing anisotropic B-factor parameters *OEGetAnisou* and *OESetAnisou* have been expanded for easier use in wrapped languages. The predicate *OEHasAnisou* and the container object *OEAnisoUij* have been added to support this functionality.
- The ability to request bond stereomark corrections to reduce the number of stereo-center correction warnings when reading MDL format files has been added. For more information, see *OEMDLStereoFromBondStereo* function, *OEIFlavor.SDF.FixBondMarks*, *OEIFlavor.MDL.FixBondMarks*, and *OEIFlavor.RDF.FixBondMarks* constants.
- 41 new ring templates have been added to the built-in ring dictionary.

Table 3: **Example of the new 2D ring templates**

Carbon Skeleton


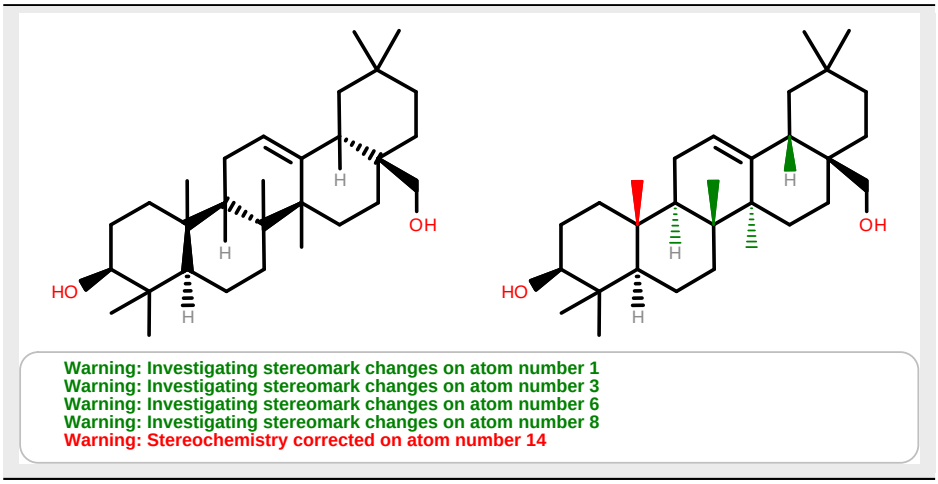
Major bug fixes

- *OEMCSType.Approximate* method for calculating the *maximum-common-substructure* of two molecules will no longer leak memory. This memory leak was introduced in the 2015.Oct release.
- The following functions and methods will no longer crash when used after *oemolithread.open* or *oemolothread.open*, but will fail and return `false`:
  - *OEReadMolecule*
  - *OEWriteMolecule*
  - *oemolthreadbase.PutMol*
  - *oemolthreadbase.GetMol*
  - *oemolthreadbase.PeekMol*

Minor bug fixes

- The following problems have been fixed for reading V3000 MDL files:
  - *OEReadMDLQueryFile* function has been generalized to more reliably identify a V3000 header and dispatch to the correct reader.
  - A parsing error for V3000 MDL files with an atom list has been fixed.
  - An issue with atom lists in V3000 format that resulted in the export of invalid atom symbols has been fixed. Atom lists will now export as "L" but will lose the original atom list information. Query features are not yet supported for molecule writing.
- A new warning will be issued for any V3000 MDL files that contain SCSR template information. This information is currently ignored.
- A new warning will be issued for any MDL files that contain Sgroup information or Rgroup definition. This information is currently ignored.
- An issue that occurred when valid cis/trans bond stereo were being converted to `double either` bonds in MDL reaction files has been corrected. This problem was introduced in the 2015.Oct release.

- The function to access anisotropic B-factor parameters *OEGetAnisou* now sets all the parameters to `0` if no data is associated with the given atom, rather than leaving the parameters in an uninitialized state. The new predicate *OEHasAnisou* can be used to determine if an atom has these parameters.
- *OEMCSSearch* constructor no longer throws a warning when initialized with a molecule that has up-to-date ring perception information.
- *OEAssignAromaticFlags* function now invalidates the hybridization perception flag since different aromaticity models affect the recorded hybridization values.
- Atom stereochemistry is more thoroughly examined and may result in multiple marks to the stereocenter(s) and/or marks to ring bond(s) for *OEMDLStereoFromBondStereo* activity to reduce the number of `Stereochemistry corrected ...` warnings on re-read of the exported structure.



Documentation changes

- Minor corrections to the *OEPerceived* namespace have been made.

5.61.2 OEBio 2.0.7

New features

- *OEPerceiveInteractionHints* function has been added to perceive possible ligand-protein interactions (contact, clash, hydrogen bond, salt-bridge, chelator, and aromatic ring stacking).

  The following classes and namespaces have been introduced to handle the new interaction types:

  - *OEClashInteractionHint* class and corresponding *OEIsClashInteractionHint* predicate
  - *OESaltBridgeInteractionHint* class and corresponding *OESaltBridgeInteractionHintType* namespace and *OEIsSaltBridgeInteractionHint* predicate
  - *OEStackingInteractionHint* class and corresponding *OEStackingInteractionHintType* namespace and *OEIsStackingInteractionHint* predicate
- *OEGetActiveSiteInteractionHintTypes* function has been added to return the interaction types perceived by the new *OEPerceiveInteractionHints* function.
- *OECrystalSymmetryParams* class has been added, along with overloaded versions of *OEGetCrystalSymmetry* and *OESetCrystalSymmetry* functions taking an *OECrystalSymmetryParams* object. This is the recommended API to use in the wrapped languages.

Minor bug fixes

- *OEMolComplexCategorizer.SetMaxPeptideResidues* method now properly sets the maximum number of residues that can be considered a peptide by *OEMolComplexCategorizer*.
- Lower-case chain IDs are now supported by *OESplitMolComplex* and related functions.
- *OERMSD* now checks whether the match stored in the given *OESequenceAlignment* object is valid prior to attempting to calculate RMSD.
- An uninitialized memory problem that occurred when *OEGetAlignment* failed to perceive an acceptable alignment has been fixed.

Deprecated OEFragmentNetwork and related API

The following warning will be thrown when using the above deprecated APIs:

> **Warning**
>
> "Warning: OEFragmentNetwork is deprecated. Use OEInteractionHintContainer instead."

Deprecated class	Replaced by
OEFragmentNetwork	*OEInteractionHintContainer*
OEFragment	*OEInteractionHintFragment*
OEFragmentConnection	*OEInteractionHint*
OEFragmentConnectionTypeBase	*OEInteractionHintTypeBase*
OEFragmentContactInteraction	*OEContactInteractionHint*
OEFragmentHBondInteraction	*OEHBondInteractionHint*
OEFragmentChelatorInteraction	*OEChelatorInteractionHint*
OEComponentTypeBase	*OEInteractionHintComponentTypeBase*
OELigandComponentType	*OELigandInteractionHintComponent*
OEProteinComponentType	*OEProteinInteractionHintComponent*

Deprecated predicate	Replaced by
OEHasConnection	*OEHasInteractionHint*
OEHasConnectionType	*OEHasInteractionHintType*
OEHasResidueConnection	*OEHasResidueInteractionHint*
OEIsChelatorInteraction	*OEIsChelatorInteractionHint*
OEIsHBondInteraction	*OEIsHBondInteractionHint*
OEIsInterConnection	*OEIsInterInteractionHint*
OEIsIntraConnection	*OEIsIntraInteractionHint*

Deprecated function	Replaced by
OEIsValidActiveSite(OEFragmentNetwork)	*OEIsValidActiveSite(OEInteractionHintContainer)*
OEGetConnections(OEMolBase,OEFragmentNet	*OEGetInteractions(OEMolBase, OEInteractionHintContainer)*
OEGetConnections(OEAtomBase,OEFragmentNe	*OEGetInteractions(OEAtomBase, OEInteractionHintContainer)*

Deprecated namespace	Replaced by
OEFragmentHBondType	`OEHBondInteractionHintType`
OEFragmentChelatorType	`OEChelatorInteractionHintType`

Documentation changes

- A link has been added from the *OEResidue* API documentation back to the *Biopolymer Residues* section to better clarify details about residue data.
- A note has been added to the `OEResidue.IsHetAtom` documentation concerning heterogen atoms being distinguished by `HETATM` records in a PDB file.
- Java and C# versions of examples *Splitting a Macro-molecular Complex* and *Split a Macro-molecular Complex Into Fragments* have been fixed.
- *Perceive and Print Protein-Ligand Interactions* example has been updated to use the new *OEInteractionHintContainer* class and corresponding API.

5.61.3 OESystem 2.0.7

New features

- `OEGeom3DRotate` template math function can now be used with *OEHalfFloat* coordinate types.

5.61.4 OEPlatform 2.0.7

New features

- `OELIMITEDSTACK` macro define value can now be controlled from compiler flags like `-DOELIMITEDSTACK=0`. This allows projects to use the C++ header files when compiling **OpenEye** C++ code for use with **OpenEye** Java toolkits.

  For more information on compiling custom **OpenEye** C++ code for use by **OpenEye** Java toolkit projects, see https://github.com/oess/oejava_extension.

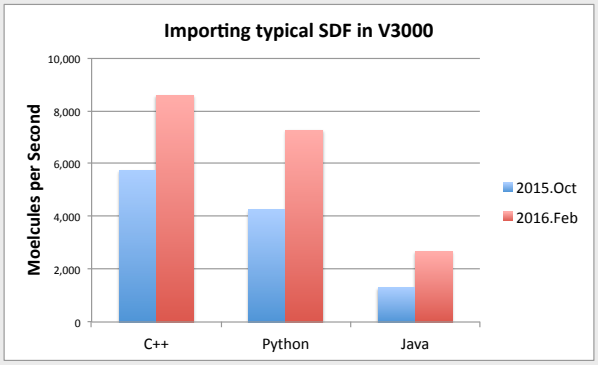
5.62 Version 2.0.6

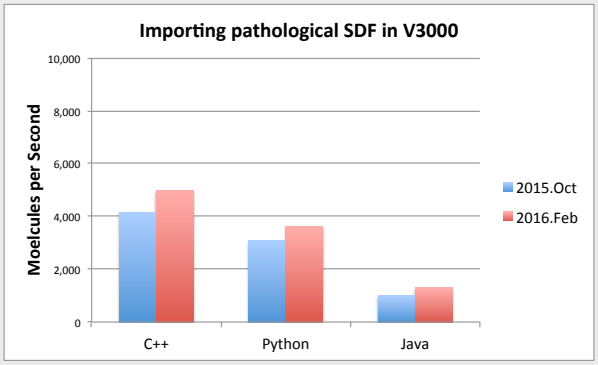
5.62.1 OEChem 2.0.6

New features

- The performance of the SDF V3000 format file parsing has been significantly improved and is now approximately 50% faster. The speed of importing an SDF file in V3000 format is now comparable to V2000. See table *Performance improvement of importing SDF V3000 file format* below that shows the improvements.

Table 4: **Performance improvement of importing SDF V3000 file format**





Note

There are some pathological cases when only a slight improvement has been achieved. From the parsing viewpoint, a pathological or atypical SDF V3000 file contains an excessive number of non-default property values, redundantly specified atom or bond properties, and/or atom coordinates in scientific notation which requires a more general but slower parsing activity. For typical SDF V3000 files, a significant improvement was seen, but generally the magnitude of the improvement is largely input data dependent.

- `OEWriteMolToString` and `OEReadMolFromString` overload functions have been added to allow specification of the file format using the *the OEFormat namespace*.
- *`OEMolBase.Clear`* performance has been improved whenever the molecule is already empty.
- The following flavors have been added to the *`OEOFlavor.MOL2`* namespace: *`OEOFlavor.MOL2.Forcefield`*, *`OEOFlavor.MOL2.ChargePrecision`*, and *`OEOFlavor.MOL2.GeneralFFFormat`*. These flavors allow writing nonstandard variants of Tripos MOL2 files targeted at the general force field community.

Major bug fixes

- *`OEMMFF94PartialCharges`* function no longer crashes with incorrect atom types.
- *`OEUniMolecularRxn`* function no longer crashes when bonds to mapped explicit hydrogens are deleted.

Minor bug fixes

- *`OESortConfsByTag`* function can now sort conformations by generic data with double type.
- *`OE2DRingDictionary.AddRings`* method now allows the addition of ring templates with extremely high average bond length. These ring templates are normalized before inserting them into the ring dictionary.
- **OEChem TK**'s MDL V2000 and V3000 readers have been improved to handle nonstandard or incorrect MDL or SDF files.
  - *`OEReadMolecule`* now warns about invalid bond stereo marks on non-single bond types and will ignore them. Additionally, a common error in CTfile format files is the presence of a "wedge either" bond on a double bond. This latter error is now automatically changed to the assumed (non-wedge) "double either" bond mark and a warning is generated.
  - *`OEReadMolecule`* now attempts to read a variant of the SDFile format that contains blank line(s) before the start of the SD data appendices. Although this is a deviation from the CTfile format, this format has been known to occur in the wild. This change now impacts the use of concatenated MOL files when the

structures have blank molecule titles. In general, concatenated MOL files are a much less preferred strategy for multiple record structure input and should be avoided. It is highly recommended to always use SDF files for multi-record input since an explicit record delimiter is always present.

  - When *OEReadMolecule* encounters a connection table format error for SDF format files, it now advances to the next record delimiter. Previously, it would have attempted to reset and reread at an arbitrary point in the corrupted file, possibly generating additional warnings.
  - *OEReadMolecule* and low level MDL format readers are now more tolerant for V3000 format files that contain arbitrary collection types. Previously, only stereo collections and highlight collections were allowed. Now a message about unknown types generates the warning `Skipping unknown collection type, XXX/YYY`, with `XXX/YYY` indicating the specific collection type that was ignored. Unknown collection information is not persisted to any output format types: it is well and truly skipped!
  - A warning is now thrown when multiple rgroup label sites (e.g., R1R2) are encountered, indicating that this type of representation is not yet supported.
  - When reading V3000 format containing pseudo-atoms (i.e., atoms not in the internal **OpenEye** element list), the atom symbol information is no longer lost but can be retrieved from *OEAtomBase.GetName* and is now the same as V2000 file format handling.

- When an *OEMolBaseType.OEMiniMol* molecule implementation is instantiated from another *OEMolBase* instance that contains one or more atoms, the dimension code is also copied so that *OEMolBase.GetDimension* matches the dimension setting from the original *OEMolBase* instance.
- *OEReadMolecule* is now more tolerant for SKC format files containing explicit string tags of 0-length.
- *OEAssignHybridization* now ignores transition metals, lanthanides, and actinides and sets their hybridization to *OEHybridization.Unknown*. As a result, these atoms are no longer inadvertently considered to be potential tetrahedral stereocenters.
- *OEMolDatabase* cannot support file formats without explicit record delimiters, so files such as MOL, MDL, and RXN cannot be supported. A properly formatted SDF file is the preferred input to initialize the *OEMolDatabase* class.
- *OEMolDatabase* now fails early and refuses to parse the junk data when a file changes underneath an *OEMolDatabase*. This can happen when an NFS client changes a file that is already open on another NFS directory, invalidating the NFS client that is using *OEMolDatabase*.
- *OESweepRotorCompressHydrogens* no longer returns `false` when the molecule passed in does not contain any hydrogens. However, it returns `false` if the molecule contains any deleted atoms, as it is then likely that the rotor compression data is already corrupted.
- *OEGeom3DMatrixInvert* function has been fixed.
- A bug that allowed the `!DEFAULT` value of a parameter in the configuration file to be set to a value that is illegal for that parameter has been fixed. A warning is now thrown when an illegal value is set.
- A bug that caused PDB Data records, such as `REMARK` and `SSBON`, to be clipped at 72 characters instead of 80 characters has been fixed.

C++-specific changes

- A const *OEGraphMol*, *OEMol*, or *OEQMol* no longer results in a compilation error when using the following generic data getters that should have been previously marked `const` in the header file: *GetBoolData*, *GetIntData*, *GetFloatData*, *GetDoubleData*, and *GetStringData*.
- *OEPRECompress* that outputs an *OEMol* has been deprecated in favor of the `OEPrepareFastROCSMol` function. This function will be removed in a future release.

> **Note**
>
> *OEPRECompress* that alters the binary IO handlers on an *oemolstreambase* is still supported and is the appropriate way to turn on PRE-compressed OEB.

Python-specific changes

- Passing None to the *oemolistream* and *oemolostream* constructors no longer results in a crash.

Documentation changes

- *OEDeleteEverythingExceptTheFirstLargestComponent* function is now documented.

5.62.2 OEBio 2.0.6

New features

- *OESplitMolComplexOptions.SetSplitCovalentCofactors* and *OESplitMolComplexOptions.GetSplitCovalentCofactors* methods have been added to control the splitting of covalent cofactors from a macromolecular complex. The new constant *OESplitMolComplexSetup.CovCofactor* controls whether *OEConfigureSplitMolComplexOptions* sets up command-line parsing for this option.
- *OEMolComplexCategorizer* can now recognize a multi-residue *OEAtomBondSet* as a covalently attached ligand or cofactor.
- *AminoAcid*, a new residue database category, has been introduced. It consists of standard amino acids and common variants such as seleno-methionine. Previously, these had been listed in the category *Cofactor*.
- *OESplitMolComplexOptions.SetWarnNoLigand* and *OESplitMolComplexOptions.GetWarnNoLigand* methods have been added to control whether the molecular complex splitting functions generate a verbose message warning whenever a ligand is not identified.
- *OEClearMolComplexSDData* function has been added to remove SD tags generated by *OEGetMolComplexComponents*.
- *OEGetAlignments* has been added to deal with multiple chains in each structure. The method returns an iterator of alignments, one for each pairwise chain alignment. *OEGetAlignment* now returns the highest scored alignment from *OEGetAlignments*. *OEGetSimpleAlignment* is a replacement for *OEGetAlignment*, which only looks at the first chain in each sequence.
- *OEWriteAlignment* has gained a third parameter to allow varying the width of the output.
- The following methods have been added to the *OESequenceAlignment* class:
  - *OESequenceAlignment.GetChainID*
  - *OESequenceAlignment.GetSequenceIdentity*
  - *OESequenceAlignment.GetSequenceSimilarity*.

Minor bug fixes

- Titles generated by functions *OESplitMolComplex* and *OEGetMolComplexComponents* no longer contain single quotes or blank characters.
- Ongoing maintenance has been performed in the *OEResidueCategoryData* database used by *OESplitMolComplexOptions*. Residues have been removed from the *Polymer* and *Misc* lists.

Documentation changes

- The examples have been updated to perceive residues when this is not performed by the default molecule reader activity (for example, in the case of `.mol2`). With this update, examples that had previously been transforming input hydrogen names from a PDB file to the new nomenclature (`closecontacts`, `makealpha`, `subsetres`, and `swapaieres`) now retain the input hydrogen names.

5.62.3 OESystem 2.0.6

New features

- *OEThreadedDots* has been added to provide a thread-safe way to output progress bar dots to the terminal when the work is being updated from multiple threads simultaneously.

5.62.4 OEPlatform 2.0.6

- Minor internal improvements have been made.

5.62.5 OEGrid 1.5.3

Major bug fixes

- `OEMakeGridFromCenterAndExtents` now propagates errors with the grid construction.
- `OEReadGrid` can now read a gzipped *OESystem::OEFizzGrid*.

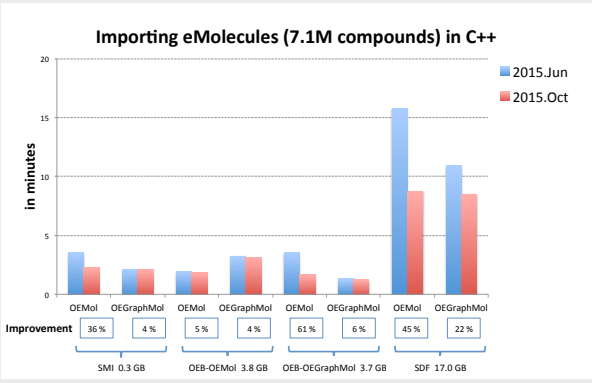
5.63 Version 2.0.5

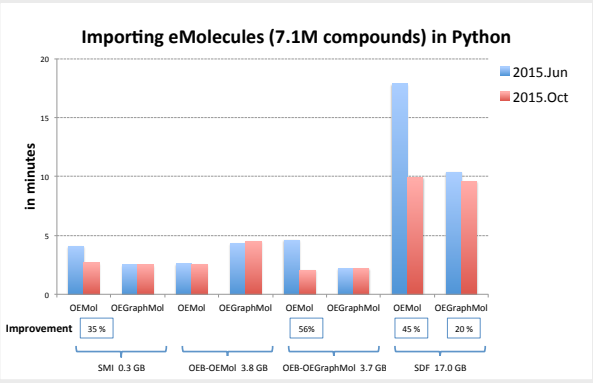
5.63.1 OEChem 2.0.5

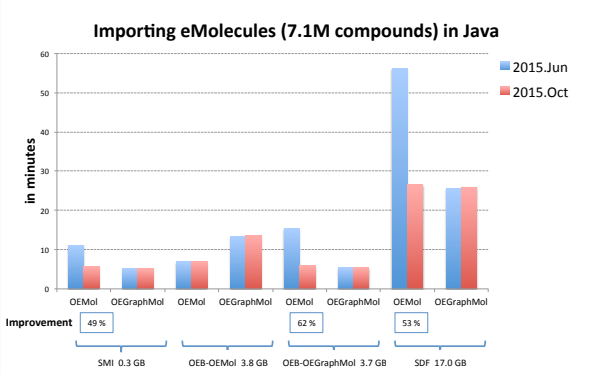
New features

- `OEReadMolecule` into an *OEMol* is now equivalent to reading into an *OEGraphMol* for the majority of use cases. To achieve this, it is now supported to call `OEReadMolecule` on *OEConfBase* as returned by `OEMCMolBase.GetActive`. See table *Performance improvement of importing molecules from various file formats* below that shows the improvements by supported languages.

Table 5: **Performance improvement of importing molecules from various file formats**





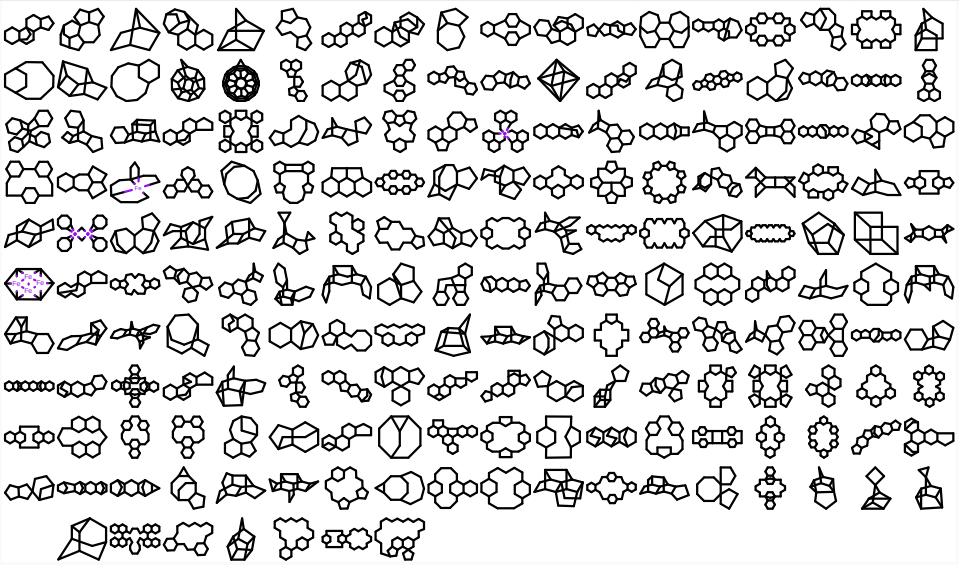
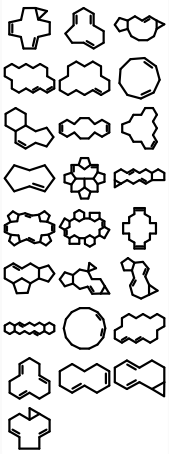


Note

- OEB-OEMol - **OpenEye** OEBinary file format storing molecules as *OEMol* (multi-conformer molecule representation)
- OEB-OEGraphMol - **OpenEye** OEBinary file format storing molecules as *OEGraphMol* (single-conformer molecule representation)

- The performance of reading V2000 MDL MOL files has been significantly improved. See table *Performance improvement of importing molecules from various file formats* above.
- The performance of 2D coordinate generation has been improved by as much as 33% in some cases.
- `OEIsMember.Add` and `OEIsMemberPtr.Add` methods have been added.
- Added new `OESMILESFlag.AllBonds` output Flavor for SMILES generation. When set all bonds will be explicit in the generated output SMILES.
- A new `OEIsSDDataFormat` function that can return whether the given file format supports SD data information has been added.
- `OEGetMDLDimensionFromCoords` and *OESetMDLDimensionFromCoords* function have been added.
- *OEMMFFTypeResult* class was added along with a new `OEMMFFAtomType` overloaded function that provides better support for the wrapped languages (Python, Java and C#).
- 213 new ring templates have been added to the built-in ring dictionary.

Table 6: **Example of the New 2D Ring Templates**

Carbon Skeleton	Specified Cis/Trans
	

- The performance of reading *OEFormat.CDX* files has been improved.
- *SetTitle* method of *OEConfBase* class is now slightly faster.

Major bug fixes

- The memory subsystem in the **OpenEye Toolkits** has been revamped to allow more types of objects to pass safely between operating system threads by default. The following objects are no longer affected by *OESetMemPoolMode*:
  - *OEMol*
  - *OEGraphMol*
  - *OEQMol*
  - *OESubSearch*
  - *OEBase*

  The upgrade fixes crashes observed in the **OEMolProp TK** and **OEDocking TK** in multi-threaded web servers. A more thorough description of what can and can't be done with threads with the **OpenEye Toolkits** can be found in the recently updated *Multi-Threading* chapter.

  Care was taken to ensure the same or better level of performance with the new system. However, if you experience a performance degradation when switching to the **Toolkit 2015.Oct** release, please do not hesitate to contact support@eyesopen.com.
- Previously, *OEMolBase.NewAtom* would not copy coordinates when the molecule was the *OEMolBaseType.OEMiniMol* implementation. This problem has been fixed.
- *OEMCMolBase.GetMCMolTitle* will now be consistent when reading single-conformer file formats. Previously, reading SMILES and CDX files would set the title on the parent *OEMCMolBase*, but MOL2, CSV, SDF, and single-conformer OEB file formats would not. All these file formats will now initialize the parent *OEMCMolBase* title allowing more consistent behavior when annotating conformer titles (warting) across all file formats. The conformers can be warted, leaving the parent title intact as well.

- `OEChem::OEConfBase::operator bool` will now conform better to the *OEMolBase* API by returning whether the molecule contains atoms. The previous behavior was odd and unused, returning the state of the coordinates in the conformer.
- `OEChem::OEConfBase::Clear` will now be consistent with the *OEMolBase.Clear* API effectively clearing away all generic data, SD data, atoms, and bonds from the parent *OEMCMolBase*.
- *OEPerceiveResidues* will no longer crash when a thread has a small stack size and the molecule is very large.
- *OEAddMols* will no longer crash when used on *OEMolBaseType.OEMiniMol* molecule implementations.
- *OEBondBase.Clear* will now clear generic data for *OEMolBaseType.OEMiniMol* molecule implementations.
- The following changes were made to improve the support for *OEFormat.CDX* file format:
  - The problem that caused 2D coordinates to be flipped resulting in flipped atom stereo after round-tripping has been fixed.
  - Hydrogen counts for charged aromatic atoms such as `[cH-]1cccc1` have been fixed.
  - *OEBondStereo.Wavy* and *OEBondStereo.DoubleEither* bonds are now correctly handled.

Minor bug fixes

- The following methods of the *OEAtomBase* class are deprecated and will throw a warning when called:
  - *OEAtomBase.HasAlphaBetaUnsat* (use *OEHasAlphaBetaUnsat* predicate instead)
  - *OEAtomBase.GetAxial*
  - *OEAtomBase.SetAxial*
- Previously, when the function *OEParseSmiles* was called with a non-empty molecule, the hydrogen counts of the previously-existing atoms would be erroneously modified to satisfy normal valences. Now, the existing atom hydrogen counts are untouched.
- The function *OEReadMolecule* will now attempt to read malformed Rgroup atom information present in the atom block. Although an atom symbol of `R1` in the atom block may seem reasonable, it does not strictly conform to the **CTfile** specification. The *OEWriteMolecule* function will never export information in this form, but an attempt will be made to read malformed information on input.
- The function *OEReadMolecule* will no longer emit a warning when encountering the atom symbol * with CTfile format input, Instead, it will be treated as a *OEElemNo.Du* atom. This is in keeping with the return value from *OEGetAtomicNum* for * and a desire to reduce the number of warnings from *OEReadMolecule*.
- The function *OEReadMolecule* will now read Rgroup atom label information from V3000 format input. Previously this information was ignored.
- The function *OEReadMolecule* will now make a more concerted effort to read malformed DOS line endings from input (text) files. This fix was added to *oeistream*.
- Trailing white spaces are now removed from SMILES titles.
- When calling the *OE3DToInternalStereo* function, the bond stereo is perceived from 3D even if the atom stereo perception fails.
- Coordinates in MDL MOL files that cannot be parsed as a real floating point number will now be `0.0` as per the MDL specification. Previously, the string "NaN" would actually translate to the floating point `NaN` binary representation.
- *oemolistream.GetConfTest* is now `const` correct. The `const-ness` of the underlying *OEConfTestBase* is no longer disregarded.
- *oemolistream.GetFileName* will no longer return an empty string when *oemolistream* constructor was called on a real file.

- During substructure search, the warning: `OESubSearch::SingleMatch() is unable to match bond stereo in the target for patterns with bond stereo, call OEPrepareSearch on the target first` was sometimes incorrectly thrown. The underlying stereo perception has been revamped.
- *OESubSearch* no longer throws the following message in case when *OEPrepareSearch* is invoked on the target molecule. `OESubSearch::SingleMatch() is unable to match hybridization in the target for patterns with set hybridization, call OEPrepareSearch on the target first.`
- The *OEUniMolecularRxn* has been improved to reduce the number of odd valence and broken aromatic ring results from transformations.
- *OEPerceiveChiral* now sets the relevant perception flag for empty molecules.
- *OEIFlavor.SMI.Strict* and *OEIFlavor.SMI.Canon* flavor combination is now valid for the *OEFormat.SMI* file format.
- *OE3DToAtomStereo* and *OE3DToBondStereo* functions remove the stereo property of an atom (or bond), respectively, if it is not chiral or if its stereo configuration can not be determined from the 3D.
- *OEAtomBase.SetStereo* and *OEBondBase.SetStereo* methods now remove the stereo generic data from the atoms (or bonds) when calling with the *OEAtomStereo.Undefined* or *OEBondStereo.Undefined* value, respectively.
- *OEHasAtomStereoHydrogens* was fixed, it returns `true` for a tetrahedral atom with specified stereo and one explicit hydrogen neighbor.
- *OEHasBondStereoHydrogens* was fixed, it returns `true` for an atom that belongs to a cis/trans double bond with one explicit hydrogen neighbor.
- *OEReadOEBFile* can now be successfully used to read and round-trip an empty molecule (no atoms) to an OEB file. *OEReadMolecule* will still automatically skip molecules with no atoms.
- *OEBReadDataLength* now takes a default parameter for the maximum length.
- *OEFormat.Default* now has the same numerical value as *OEFormat.SMI* instead of *OEFormat.ISM*. As of **OEChem TK** 2.0.0 (the 2014.Feb release), *OEFormat.SMI* and *OEFormat.ISM* are functionally equivalent in **OEChem TK's** file handling, so this is mostly a cosmetic change for consistency in that effort.
- Explicit hydrogens created by parsing stereo in SMILES strings is now capped to a maximum of eight explicit hydrogens. Previously, a SMILES such as `[C@@H1000000000]` would seem to make **OEChem TK** hang indefinitely as it tried to create all those explicit hydrogens.
- *OEMaskGridByCoordsAndRadii* has been corrected to fix a bug introduced in OEGrid 1.5.0 that produced grids of the wrong size.
- The function *OEReadMolecule* will now simply ignore the V3000 highlight collection information that was previously causing a read error.
- Previously, the function *OEWriteMolecule* could emit lines exceeding 80 chars in V3000 format for structures containing many stereocenters. This is in violation of the **CTfile** specification and has been corrected.
- An obscure issue causing a crash in *OESubSearch* or *OEQMolBase.BuildExpressions* for imines and related queries with cis/trans parity and explicit hydrogens has been corrected, e.g., `[H]/N=C/C`.

Documentation changes

- *rmsd* example has been added to generate RMSD alignment for multi-conformer molecules.
- Links to the **CTfile** format document from Biovia (previously Accelrys, Inc.) has been updated. Note that registration is required to download this document.
- *OE2DRingDictionary.AddRings* return value is now documented.

5.63.2 OEBio 2.0.5

New features

- *OEBio::OEFragmentNetwork* class is now exposed in order to allow the investigation of the protein-ligand interactions perceived by **OEDocking TK** and visualized by **Grapheme TK**. The current API is **read only** - *i.e.*, the user cannot build a fragment network.
  - The fragment network (*OEBio::OEFragmentNetwork*) is a container of typed molecules. The following classes are available to classify molecules in a fragment network:
    - *OEBio::OEComponentTypeBase* abstract class
    - *OEBio::OEProteinComponentType* concrete type that identifies protein molecule of complexes.
    - *OEBio::OELigandComponentType* concrete type that identifies ligand molecule of complexes.
  - A fragment (*OEBio::OEFragment*) is a set of atoms of a molecule that is stored in the fragment network.
  - A fragment connection (*OEBio::OEFragmentConnection*) is a typed link between two fragments of the network. The following classes have been added to handle interactions perceived by **OEDocking TK** (see also `OEAddDockingInteractions` function):
    - *OEBio::OEFragmentConnectionTypeBase* abstract class
    - *OEBio::OEFragmentContactInteraction*
    - *OEBio::OEFragmentHBondInteraction* along with the related *OEBio::OEFragmentHBondType* namespace
    - *OEBio::OEFragmentChelatorInteraction* along with the related *OEBio::OEFragmentChelatorType* namespace
  - The following fragment connection predicates have been added:
    - *OEBio::OEHasConnectionType*
    - *OEBio::OEHasConnection*
    - *OEBio::OEHasResidueConnection*
    - *OEBio::OEIsChelatorInteraction*
    - *OEBio::OEIsHBondInteraction*
    - *OEBio::OEIsInterConnection*
    - *OEBio::OEIsIntraConnection*
  - The following functions have been added:
    - *OEIsValidActiveSite* function
    - *OEBio::OEGetConnections* function
- The database of non-ligand residue types maintained by the *OEResidueCategoryData* object contained within an *OESplitMolComplexOptions* has been curated. Only about 40 percent of the entries previously listed under the category *OEMolComplexFilterCategory.Misc* remain, the remaining entries were recognized to be obscure ligands or were moved to other categories, such as *OEMolComplexFilterCategory.CofactorAndLigand*. The category *OEMolComplexFilterCategory.DNA_RNA* has been expanded to include the more obscure RNA bases.
- Setting modelNum to 0 in the *OESplitMolComplexOptions.ResetFilters* method of *OESplitMolComplexOptions* now selects all models.

- Protein Data Bank *biological assembly* files, with extensions like `.pdb1.gz`, can now be read. These files use model numbers to refer to each part of an assembly (model numbers are also used to distinguish NMR models). By default, each model will be loaded as a separate molecule. See *splitmolcomplex* for an example of how to remove the `ENDM` flavor in order to load all models from a Protein Data Bank file into a single molecule.

Minor bug fixes

- A problem that may have affected which residues were selected when the option method *`OESplitMolComplexOptions.SetSeparateResidues`* was used to select binding site residues has been fixed.

Documentation changes

- *printinteractions* example has been added to list protein-ligand interactions perceived by **OEDocking TK**.
- The *Stereochemistry Perception* chapter has been rewritten for the sake of clarity.
- Examples *splitmolcomplex* and *splitmolcomplexfrags* have been updated to support the reading of multiple models from a `PDB` file.

5.63.3 OESystem 2.0.5

New features

- *`OERandom.GetSeed`* has been added to return the current state of the *OERandom* class enabling the restart of the random number generator with the given seed.
- *`OEGridSizeMultiply`* has been added to safely multiply grid dimensions together while ensuring that it will not cause the value to overflow the integer type.
- *OEOwnedPtr* is now move-constructable for any C++11 enabled compiler. OpenEye's first foray into supporting C++11 directly in our APIs. C++11 will eventually provide broader usage of smart pointers across the **OpenEye Toolkits** API.

Minor bug fixes

- *OEAnnotation* can now be successfully round-tripped through OEB.

Python-specific changes

- `HasUnsignedInt` and `GetUnsignedInt` methods have been added to the *OEInterface* class Python wrapping to allow accessing command line parameters with `!TYPE unsigned`.

C#-specific changes

- `HasUnsignedInt` and `GetUnsignedInt` methods have been added to the *OEInterface* class C# wrapping to allow accessing command line parameters with `!TYPE unsigned`.

5.63.4 OEPlatform 2.0.5

New features

- *`OEGetAbsolutePath`* is a new function that returns the full absolute path of a file.
- *`OEThread.SetStackSize`* has been added to allow controlling the maximum amount of stack size that the operating system thread will have when it is started with *`OEThread.Start`*.
- If a toolkit program fails due to a license error, the paths of any license file(s) actually read will be output for diagnostic purposes.

Major bug fixes

- `OEPlatform::OEMalloca` will no longer cause crashes when attempting to allocate amounts of memory near the maximum allowable `size_t`. `std::bad_alloc` will now be thrown instead in these situations. Since `OEPlatform::OEMalloca` is actually a C-preprocessor macro, this change may result in user code throwing additional compiler warnings when signed integer types like 'int' are passed to `OEPlatform::OEMalloca`. We recommend that users change this code to use `size_t` to avoid these warnings.

Minor bug fixes

- `oepstream` class has been removed since it never worked as originally intended.
- *`OEUncompress`* and *`OECompress`* can now take `size_t` instead of `unsigned int` allowing use on data in excess of 4GB on 64-bit machines.

Documentation changes

- *OEOnce* example code in the documentation will now actually compile. Previously, the code would actually crash prior to the *OESubSearch* memory pool refactoring of this release, if any other thread besides the `main` thread called the function first.

C++-specific changes

- **OpenEye Toolkits** header files should no longer contain the C++11 deprecated "register" keyword causing compilations with `-Werror` to fail.
- *oestream* and derived classes will no longer allow implicit conversion to `int` and `unsigned int`. This avoids surprising behavior like the following from compiling:

```
oeisstream is1;
oeisstream is2;
bool what = (is1 == is2);
```

5.63.5 OEGrid

New features

- **OEGrid TK** now features more efficient use of memory while reading grids from files. Previously, several unnecessary memory allocations had sometimes been performed during the read process.
- **OEGrid TK** grids are now officially capped to 4GB of memory since the API is designed around `unsigned int`.

Major bug fixes

- *OEScalarGrid* and all other grid types will no longer crash when the dimensions would cause overflow of an `unsigned int`. The grid types will now throw `std::bad_alloc` if attempting to make the grid memory larger than 4GB.

5.64 Version 2.0.4

5.64.1 OEChem 2.0.4

New features

- The following API have been added to generate and use 2D ring dictionaries:
  - *OE2DRingDictionary* class

- *OEIsValid2DRingDictionary* function
- *OEWrite2DRingDictionary* function
- *OEInit2DRingDictionary* function

The following sections provide examples to generate and utilize user-defined 2D ring dictionaries:

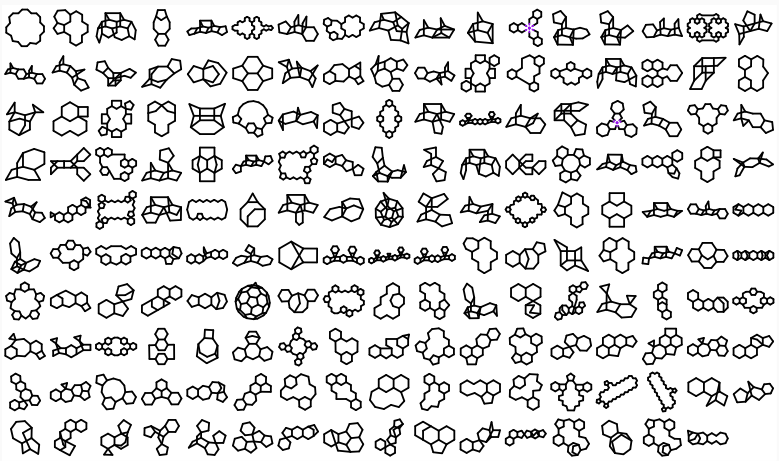
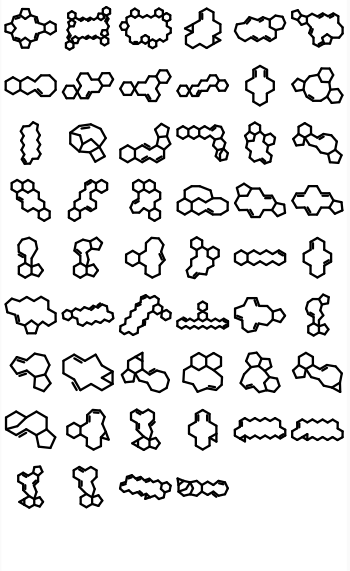
- *Create 2D ring dictionary*
- *Append 2D ring dictionary*
- *Generate 2D coordinates with user-defined ring templates*

See also

ringdict2pdf example in **OEDepict TK** will generate a multi-page PDF report of a 2D ring dictionary.

- 221 new ring templates have been added to the built-in ring dictionary.

Table 7: **Example of the New 2D Ring Templates**

Pure Carbon Skeleton Ring Templates	**Ring Templates with Specified CIS/Trans Bonds**
	

- *OEIFlavor.MOL2.Forcefield* flavor has been added to the MOL2 reader. This allows atom names in the MOL2 file to be interpreted as chemical elements while not using the second character if capitalized; for example, CA and CD are both carbon instead of calcium and cadmium. This will help reading *OEFormat.MOL2* files that contain forcefield information like partial charges.
- *OEAddCustomFASTAResidue* now provides the ability to customize *OEFormat.FASTA* input files with unnatural nucleic acids. *OEIFlavor.FASTA.CustomResidues* flavor is used to activate the new feature. *OEIFlavor.FASTA.EmbeddedSMILES* flavor has also been added to experiment with adding cyclizations to peptides using custom tethers. These features are currently considered experimental; please send feedback to support@eyesopen.com.
- *OEOFlavor.SDF.UnsetBad2DStereo* and *OEOFlavor.CDX.UnsetBad2DStereo*, new output flavors for SDF and CDX output formats, have been added. These constants control behavior when generating 2D coordinates

during output. When used, the writer will set the stereo to "undefined" for any misleading stereochemistry that may result from layout limitations.

- *OESubsetMol* has a new overload that supports the use of *OEAtomBondSet* input. This can offer significant performance improvements over the predicate overloads since a full molecule iteration can be avoided.
- *OEMolBase.ClearCoords* has been added to fully clear molecule coordinates, clear the dimension code, and reset perception flags.
- *OEMolDatabase.WriteMolecule* has been added to automatically extract a molecule from the database and write it to the *oemolostream*. This function is optimized for performance to simply move bytes around if the input *OEMolDatabase* format and the output format match exactly.
- *OEMolDatabase.GetMoleculeString* has been added to allow direct access to the molecular record data.
- *OEPRECompress* and *OESweepRotorCompressHydrogens* have been added and documented to allow advanced users to start experimenting with perfect-rotor-encoding OEB files to achieve higher rates of conformer compression. This feature is currently only available to the C++ toolkits.
- *OEMDLGetValence* has been added to provide direct access to the MDL valence model. See the *MDL Valence Model* section for more information.
- The following free functions have been added for handling dummy atoms in forcefield calculations in special circumstances:
  - `OESetForceFieldDummyAtom`
  - `OEClearForceFieldDummyAtom`
  - `OEClearForceFieldDummyAtoms`
  - `OEIsForceFieldDummyAtom`

  Currently, dummy atoms are only allowed in **Szybki TK** for non-bonded interactions.
- *OEAssignRadii* free function has been added to allow for a central entry point for any **OpenEye Toolkit** atomic radii assignment. This free function uses the *OERadiiType* namespace for choosing the type of radii to assign.

Major bug fixes

- *OEReadMolecule* will now correctly handle right-justified atomic symbols in MDL molfiles. These files are being generated by the RCSB PDB custom SDF file writer.
- *OEPerceiveCIPStereo*, the Cahn-Ingold-Prelog stereochemistry perception algorithm, has been rewritten to solve the following issues:
  - Performance issues for large molecules have been resolved.
  - Incorrect chirality perception for certain isotopic molecules, detailed in the blog post R or S? Let's Vote, has been corrected.
  - Incorrect chirality perception for stereocenters with both a hydrogen and either a deuterium or tritium attached has been fixed.
  - Incorrect chirality perception for stereocenters with both a hydrogen and a *atom (atomic number 0) has been fixed.
- *oemolistream.Setgz* with a `false` argument called after a stream has finished reading molecules will no longer cause *OEReadMolecule* to crash on subsequent calls. It will now return `false` to indicate that the stream has finished.
- *OEMolDatabase.Open* will now fail and throw a warning message if there is no disk space left for uncompressing compressed input files. In earlier releases, the operation would appear to succeed and lead to corrupted molecules past a certain point. Furthermore, the directory that *OEMolDatabase* uses for temporary files can now

be controlled through the TEMP or TMP environment variables on Windows or the TMPDIR environment variable on Linux and OSX.

- *OEMol* will now work with the V3000 file format much more reliably. This was accomplished by making sure that multi-conformer molecules can handle groups appropriately. The *OEGroupBase* class is a container of atom and bond pointers inside an *OEMCMolBase*. The *OEGroupBase* class is currently used to store MDL-enhanced stereo information. Each conformer of the multi-conformer molecule has its own separate container. Now methods such as *OEMolBase.GetGroups* will provide access to the groups of the active conformation.

Minor bug fixes

- *OEFormat.OEB* format reading will no longer cause a rare crash when dealing with large amounts of SD data, PDB data, or bond integer types.
- *OEMatchBase* now checks parameters to the overloaded versions of the *OESubsetMol* function to ensure that the given match corresponds to source and destination molecules.
- *OEMolBase.NewGroup* method signatures now take const std::vector of atom and/or bond pointers. Previously, these arguments were not const even though the vectors were not altered.
- *OEReadMolecule* will now perform chirality perception for 0D MDL molfiles to ensure that *OEAtomBase.IsChiral* and *OEBondBase.IsChiral* are correct.
- *OEGetAtomicNum* is now tolerant of leading spaces for the string argument passed in.

Java-specific changes

- *OEBinaryIOHandlerBase* had accidentally been removed from Java in the 2014.Oct release. This broke the older idiom for preserving rotor-offset compression in OEB files using *OEInitDefaultHandler*. The preferred API is to now use *OEPreserveRotCompress* directly on the molecule streams, avoiding the need to access the *OEBinaryIOHandlerBase* internals entirely.
- The OEUnaryRoleSetPredicateFunc function was renamed to OEUnaryRoleSetBoolFunc for the sake of consistency.
- *OERoleSet* predicates can now be created and customized in the same way as atom and bond predicates. *OERoleSet* predicates are an important part of customizing the new **OEBio TK** *Splitting Macromolecular Complexes* feature.
- *OERoles* class has been added as a concrete implementation of the *OERoleSet* abstract base class.

Documentation changes

- *2D Coordinate Generation* chapter has been added.

5.64.2 OEBio 2.0.4

New features

- An API has been added for classifying and separating the various components of a macromolecular complex by their functional roles. See the *Splitting Macromolecular Complexes* section for more information.
  - *OESplitMolComplex* function
  - *OEGetMolComplexComponents* function
  - *OECountMolComplexSites* function
  - *OESplitMolComplexOptions* class
  - *OEMolComplexCategorizer* class

  - *OEResidueCategoryData* class
  - *OEMolComplexFilter* class

- `OEGetResidues` function has been added to allow iteration over all unique *OEResidue* objects in an *OEMolBase*.

Documentation changes

- *Protein Preparation* chapter has been added to document the APIs for splitting macromolecular complexes into their components.

5.64.3 OESystem 2.0.4

New features

- `OEThrow.SetLevel` and `OEThrow.GetLevel` race conditions should be largely alleviated now. `OEErrorHandler.SetLevel` and `OEErrorHandler.GetLevel` will now affect the error message level for the current thread only; all new threads launch with the default error level of `OEErrorLevel.Default` (`Info` or above).

> **Warning**
>
> Multi-threaded **OpenEye Toolkit** programs may need to be updated to allow the desired output to be sent to standard error. However, the previous behavior was race-condition prone and difficult to get correct.

Minor bug fixes

- *OERandom* constructor will no longer take an `unsigned int` as input. Previously, the `unsigned int` would be implicitly cast to `bool` and would control whether to time seed the constructor. However, most users actually intend to use an `unsigned int` as a random number seed meant for the `OERandom.Seed` method. This change should catch this common mistake at compile time.

Documentation changes

- *OEMath Functions* chapter now contains more detailed descriptions of the functions.
- Documentation has been added for the following classes:
  - *OEBinaryIOHandlerBase*
  - *OEBinaryIOHandler*
  - *OEBInitializer*
  - *OEBoundedBuffer*
  - *OEOwnedPtr*
  - *OEProtectedBuffer*
  - *OEUnboundedBuffer*

5.64.4 OEPlatform 2.0.4

New features

- `OEFileTempPath` will now use the directory specified by the `TMPDIR` environment variable on Linux and OSX. This allows the user to control where **OpenEye Toolkits** will create temporary files. This is especially useful since *OEMolDatabase* can use a lot of temporary storage when dealing with compressed data files.

5.64.5 OEGrid 1.5.1

- Minor internal changes have been made.

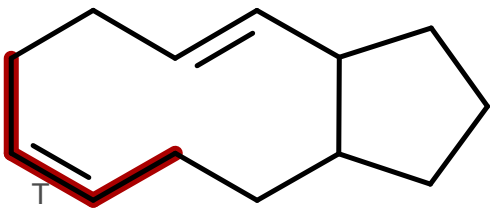
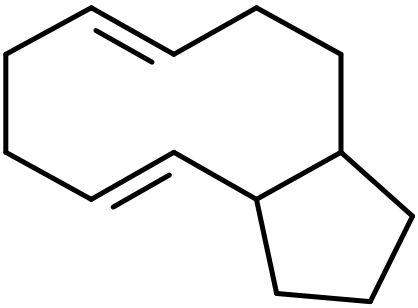
5.65 Version 2.0.3

5.65.1 OEChem 2.0.3

New features

- *OEGenerate2DCoordinates* added the following new features to improve the 2D layout of molecules:
  - The ring template dictionary search algorithm now has the ability to preserve cis/trans bond stereo. The 2014.Feb and 2014.Jun releases contained a bug that caused bond stereo to be erroneously inverted. The 2014.Oct release prevented the erroneous inversion by disallowing creation of such 2D coordinate layouts. This release is the next step in improving 2D coordinate generation for large macrocycles: ring templates with matching cis/trans stereo chemistry can now be used.
  - 252 new ring templates have been added:
    - 180 complex ring systems with anonymized carbon frameworks
    - 72 common macrocycle ring systems with specified cis/trans bond configuration

Table 8: **Example of 2D coordinate generation for macrocycles with cis/trans bond stereo**

OEChem TK 2.0.2	OEChem TK 2.0.3
C1CC2CC/C=C/CC/C=C/C2C1 	C1CC2CC/C=C/CC/C=C/C2C1 

  - A new *OEExtractRingTemplateOptions* class and *OEExtractRingTemplates* function, which are used to extract potential new ring templates, have been added.
  - A new **OEChem TK** example that extracts ring templates for 2D coordinate generation has been added. For more information, see the *Extract 2D ring templates* section.

> **Note**
>
> To improve the depiction of the ring systems in a collection of molecules, please run the *Extract 2D ring templates* script on the collection and send the results to support@eyesopen.com. **OpenEye** will be happy to include the ring systems in the next release of the toolkits.

- A new *OEIs2DFormat* function that can determine whether the given file format supports 2D coordinate information has been added.
- Canonical SMILES generated with the *OESMILESFlag.SuperAtoms* flag will now respect any value that is placed in the *OEAtomBase.GetName* field while canonical ordering the molecule. This feature is useful for creating specialized SMILES that can be canonicalized and deduplicated.
- A new *OEReadOEBFile* reader has been added that makes low-level OEB parsing available to Python, Java, and C#. Note: this function is considered advanced; *OEReadMolecule* is still preferred.

Major bug fixes

- InChI creation will now fail if the molecule being converted cannot be handled by the InChI reference implementation. The **OEChem TK** will throw a warning message about the molecular data that cannot be converted and return an empty string. This also protects against many possible crashes inside the InChI reference implementation due to abnormal molecules.
- InChI creation will no longer overflow the stack when an atom has more than 20 neighbors.
- The state of the `chiral` flag from MDL files is now correctly managed and maintained by *OEReadMolecule* and *OEWriteMolecule*. Previously, `non-chiral` (relative) stereochemistry structures were being promoted incorrectly to `chiral` (absolute) stereochemistry.
- *OESubSearch* constructors no longer create or use uninitialized memory. This bug was introduced in the 2014.Jun release and caused erroneous warning messages about explicit and implicit hydrogen mismatches.
- *OEPDBOrderAtoms* previously sorted residues improperly due to a buffer overflow for the *OEResidueIndex.YG* residue index corresponding to the T-RNA base wybutosine. This has been fixed.
- *OEWritePDBFile* will no longer buffer overflow for atoms that have their name fields set.
- *OEWriteMacroModelFile* will no longer buffer overflow for atoms that have their name fields set.
- *OELibraryGen* previously had a thread-safety issue when generating libraries. This has been fixed.

Java-specific changes

- *OEGetSDDataPairs* will again return a `OESDDataIter`. The return value changed in the 2014.Jun release from "OESDDataIter" to "OEPDBDataIter" by accident.

API changes

- The following public API points were largely undocumented and have been removed as internal implementation details, including the removal of the public header:
  - `OEFuzzy OEGetAtomicNumconst OEExprBase*,...`
  - `OEFuzzy OEGetAromatic(const OEExprBase*,...)`
  - `OEFuzzy OEGetFormalCharge(const OEExprBase*,...)`
  - `OEFuzzy OEGetImplicitHCount(const OEExprBase*,...)`
  - `OEFuzzy OEGetHCount(const OEExprBase *expr,...)`
  - `OEFuzzy OEGetBondOrder(const OEExprBase*,...)`
  - `bool OEIsSingleOrAromatic(const OEExprBase*)`
  - `OEFuzzy OEGetStereo(const OEExprBase *expr,...)`

  Additionally, the function *OEDisassembleExpressions* has been explicitly marked as deprecated in the documentation and will be removed in a future release.

Minor bug fixes

- *OEMolBase.Clear* will no longer leak memory for *OEMCMolType.OEDBMCMol* after *OEMolBase.Compress* has been called.
- A warning is now thrown if kekulization is skipped and *OEParseSmiles* subsequently finds a bond with a bond order of five.
- Warnings will now be issued for enhanced stereo collection in V3000 format files that have atoms in more than one enhanced stereo collection. A given stereo center atom can be in no more than one stereo collection type.
- Enhanced stereo collection information in V3000 MDL files is now correctly parsed for multiline appendix information.
- Parsing of V3000 stereo collection information has been corrected to combine explicitly named stereo collections. Previously, multiple stereo collection entries of the same name in separate appendix entries generated different stereo groups.
- For V3000 MDL files, implicitly defined stereo collection information is now explicitly exported from *OEWriteMolecule*. Defined stereocenters that are not included in the explicit enhanced stereo collections will be added to the appropriate stereo collection based on the state of the `chiral` flag.
- For MDL files, the writer will always attempt to export in V2000 format to ensure maximum compatibility with third party programs. If V3000 format files are explicitly desired regardless of whether the molecule is compatible with V2000, use *OEOFlavor.MDL.MV30*.
- Undefined behavior in *OEMolBaseType.OEMiniMol* has been fixed.
- *OEMolDatabase.Save* will now throw a warning message and return `false` when attempting to save the database to the same file.
- *OEMCMolBase.NewConf* will now throw a warning message if the number of atoms in the *OEMolBase* or *OEConfBase* is different than the *OEMCMolBase* into which the conformer is being constructed. This addresses the common problem of creating conformers from molecules that simply differ in the number of implicit or explicit hydrogens. Note: *OEMCMolBase.NewConf* is still designed to be very fast, so the safety checks are intentionally kept minimal.
- *OESetRotamer* now properly returns `true` to indicate success when applied to residues that have fewer than four chi angles.

Documentation fixes

- Extensive documentation has been added to the *OEExprOpts* namespace and how it affects *OESubSearch* and *OEMCSSearch*.
- An example code has been added that describes how to retrieve an *OEGraphMol* object as a *OEInterface* parameter.
- A large warning message has been added to the *Multi-Threaded OEChem* chapter, alerting users that passing molecules between threads is not supported by default.

5.65.2 OEBio 2.0.3

New features

- A new *OEBio::OEFragmentNetwork* class that represents networks of fragments and their interactions has been added. This is being used by the new protein-ligand rendering capability in the **Grapheme TK**. This class is considered beta and will likely change over the next year as the protein-ligand rendering feature is refined.

5.65.3 OESystem 2.0.3

New features

- *OENumberToString* will now accept `long double` arguments.
- *OEIsSafeCast* has been added for determining whether a given value can be safely cast to a particular return type without loss of information.

Minor bug fixes

- `OESystem::OEBinaryRead` no longer crashes nor displays undefined behavior when there is a mismatch between the template parameter and the registered reader objects.
- *OEUniMolecularRxn* now strips internal generic atom data annotations on the generated product molecules.
- *OEUniMolecularRxn* now corrects implicit hydrogen counts on charged atoms in the product molecules.

Documentation fixes

- Function documentation for *OEParseHex* has been added.

5.65.4 OEPlatform 2.0.3

New features

- *OEGetPlatformInfo* will now generate more accurate descriptions of modern Windows operating systems.

Minor bug fixes

- *OEGetHostIdent* will no longer leak memory.

Documentation fixes

- Documentation has been added for the following functions:
  - *OEGetDomainName*
  - *OEGetHostIdent*
  - *OEGetHostName*
  - *OEGetIPAddress*
  - *OEGetNumProcessors*
  - *OEGetPlatformInfo*

5.65.5 OEGrid 1.5.0

New features

- *OEReadMTZ* will now automatically recognize column names generated by PHENIX in MTZ files.

5.66 Version 2.0.2

5.66.1 OEChem 2.0.2

New features

- *OEGenerate2DCoordinates* has an additional 550 complex ring systems that it can use while laying out coordinates. Any structures that contain one of these complex ring systems will see dramatically better 2D coordinate generation.
- `OEChem::OEMCMolBaseT` and `OEChem::OEConfBaseT` have been renamed to *OEMCMolBase* and *OEConfBase* respectively. This can only really have a technical effect on users of the direct C++ API. Python, Java, and C# users never saw a distinction between the template classes with the `T` suffix. For higher level language users, navigating the documentation and OEChem's type hierarchy should be greatly simplified.
- *OEUncolorMol* is a new function for performing tunable uncoloring strategies for an input molecule. Additional uncoloring strategies may be available in future versions.
- Significant performance improvement for *OEMolBase.GetAtoms*. Algorithms that do molecular graph traversal should see a significant performance benefit. For example, *OEShortestPath* was measured to be 25% faster.
- *OEPreserveRotCompress* added as a convenience function to ensure rotor-offset-compression is preserved when reading and writing molecules to OEB files. This function should be called on any *oemolistream* object that should preserve the rotor-offset-compression data needed to reproduce a smaller OEB file.
- The performance of reading and writing rotor-offset-compression files has been improved, both when rotor-offset-compression data is preserved, and when it is not.
- Multiple lines of a CSV file format can be read into successive conformers of a *OEMCMolBase* object. The SD data representing the data in the CSV file will be attached to each conformer.
- *OEAtomBondSet* added the following new methods to support the addition and removal of atom and bond object from the set:
  - *OEAtomBondSet.Add*
  - *OEAtomBondSet.ClearAtoms*
  - *OEAtomBondSet.ClearBonds*
  - *OEAtomBondSet.ClearAtomsAndBonds*
  - *OEAtomBondSet.Remove*
  - *OEAtomBondSet.RemoveAtom*
  - *OEAtomBondSet.RemoveBond*

  Adding of objects now verifies the membership of the object in the set prior to addition which will incur slightly more overhead than previous versions. This membership checking is in keeping with the spirit of uniqueness for a "set" implementation.
- *OESubsetMol* has new overloads to support use of subsetting activities using the *OEAtomBondSet* object to define the subset.
- *OEMDLPerceiveBondStereo* will cause *OEReadMolecule* to produce fewer warning messages. *OEMDLPerceiveBondStereo* was creating ambiguous stereo bond marks, additional attempts are now made to apply unambiguous stereo bond marks. In some cases, this may result in >1 bond mark to a center to fulfill the requirements of the internal checking algorithm for validly marked centers. Pathological geometries remain where no valid marks can unambiguously define the stereocenter. For example, for some fused ring systems, the required (perhaps multiple) stereomarks would result in unpalatable marked stereocenters.

  Hint

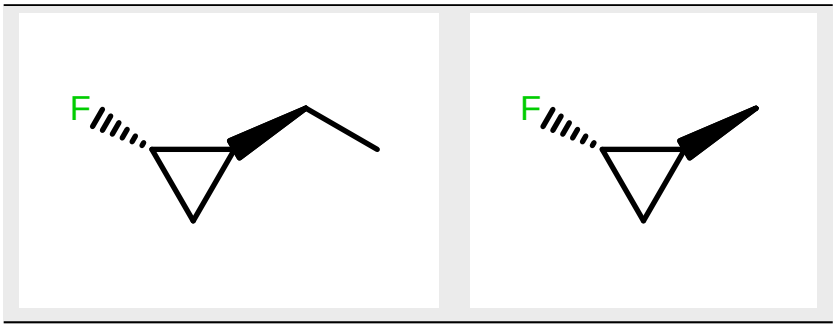
  For some cases, *OEMDLPerceiveBondStereo* will **not** be able to apply marks that will satisfy the *OEReadMolecule* perception, and a warning message will be thrown by *OEReadMolecule* indicating that

these should be redrawn or have coordinates regenerated, e.g., adding an explicit hydrogen at the bridgehead(s) to carry a stereomark.

Note

OEWriteMolecule automatically calls *OEMDLPerceiveBondStereo* when writing to the *OEFormat.MDL* and *OEFormat.SDF* file formats.

- *OEMDLHasIncorrectBondStereo* has been modified to check additional stereo environments for potential errors. For example, both example structures below are now identified as having incorrect bond stereo marks where previously only the left structure was identified.



- *OEMol* can now represent conformer coordinates in *OEHalfFloat* (16-bit), `double` (64-bit), and `long double` (>=64-bit) precision. These molecule types can be accessed through the corresponding constants in the *OEMCMolType* namespace.

  The following features were added to work in parallel with this new feature:

  - Reading and writing *OEHalfFloat* (16-bit) precision conformers to the *OEFormat.OEB* file format, including rotor-offset-compressed files.
  - Reading and writing `double` (64-bit) precision conformers to the *OEFormat.OEB* file format, including rotor-offset-compressed files.
  - Reading and writing `double` (64-bit) precision conformers to the *OEFormat.XYZ* file format. By default, the XYZ writer will introspect the molecule type to determine how many significant digits to write. *OEOFlavor.XYZ.SinglePrecision* and *OEOFlavor.XYZ.DoublePrecision* can be used to force either 5 or 18 significant digits.
  - *OEMolBase.SetCoords*, *OEMolBase.GetCoords*, and *OEMCMolBase.NewConf* can now take coordinates in *OEHalfFloat* and `long double` precision.
  - *OEConfBase.GetCoordsPtr* signature has been changed for better performance. Users should still prefer using *OEConstCoords* and *OEMutableCoords*.
  - Conformer geometry operations will now preserve the precision of the input conformer coordinates.

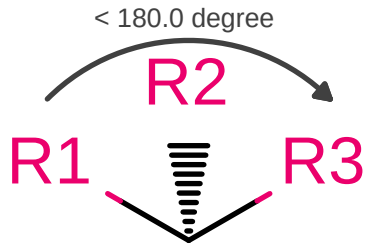
Note

These alternative precision conformer implementations are not used by default. They require the user to explicitly specify a constant from the *OEMCMolType* namespace to the *OEMol* constructor.

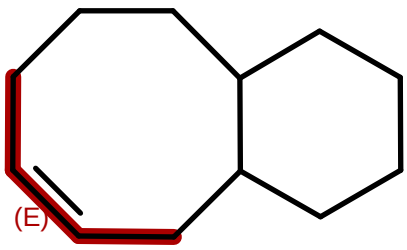
> **Warning**
>
> Older versions of OEChem, and thus many previously released OpenEye applications will not be able to read `OEB` files created from these alternative precision conformer molecules. Care should be taken to only pass these files to toolkits and applications that use at least the 2.0.2 version of OEChem. Note, this is not the default, the default is still to generate OEB files that can interoperate with older versions of OEChem.

Major bug fixes

- *OEMDLStereoFromBondStereo* and *OEMDLPerceiveBondStereo* will now generate consistent atom stereo when a wedge or hash bond occurs between atoms with an angle less than 180 degrees as shown in the following figure. This will make OEChem consistent with the official Accelrys definition of stereo in this case. As well as ensuring that SMILES can be round-tripped through an OEChem generated `SDF` file that has had 2D coordinates automatically generated.



- *OEMDLHasIncorrectBondStereo* had a major bug introduced in the 2014.Jun release. The bug was an incorrect array allocation and an out of bound access that resulted in nondeterministic processing of stereocenters and potential stack corruption. The 2014.Jun release should not be used for the *OEMDLHasIncorrectBondStereo* function.
- *OEWriteMolecule* will no longer improperly flip bond stereochemistry of complex macro-cycles when writing to *OEFormat.MDL*, *OEFormat.SDF*, or *OEFormat.CDX* file formats. This was caused by the addition of automatically generating 2D coordinates when writing to these file formats in OEChem 2.0.0, the 2014.Feb release. In cases when the generated 2D coordinates are inconsistent with the cis/trans bond stereo configuration of the molecule, the coordinates will be zeroed out, i.e., the molecule will be written with no coordinates. This can occur when generating 2D coordinates for a molecule that contains a macro-cycle with specified cis/trans stereo bond configuration like the following:



- Writing to the *OEFormat.SDF* and *OEFormat.MDL* file formats will no longer improperly promote structures with relative stereochemistry to absolute stereochemistry in some cases. This case was when an input structure had atom stereomarks and a molecule chiral flag of '0' (i.e. relative stereochemistry) erroneously causing a promotion of relative stereochemistry to absolute stereochemistry on export to *OEFormat.SDF* or *OEFormat.*

MDL. This change only affects structures that originate from the V2000 and V3000 molfile formats without 3D coordinates. Structures from SMILES and other formats don't represent relative stereochemistry, therefore, those formats will still only translate absolute stereochemistry.

- *OEMolDatabase.GetTitle* would sometimes return an empty title for multi-conformer OEB files that had their titles stored on the conformer. *OEMolDatabase.GetTitle* will now recognize this case and use the title from the first conformer if the top-level *OEMCMolBase* does not have a title.
- Rotor-offset-compressed OEB files created on big-endian machines should now be readable on little-endian machines again. This bug was introduced in the 2011.1 release, the OEChem 1.7.5 release. The only big-endian architecture supported since then was PowerPC, and that support was dropped due to lack of usage.
- *OEGetDistance2* that takes a *OEConfBase* will now return the square of the distance instead of the distance as it should. This bug was introduced in 2014.Feb, the OEChem 2.0.0 release. OEGetDistance2 functions that take arguments other than OEConfBase, e.g, OEMolBase, never had a bug introduced and will continue to return the square of the distance between the two atoms.
- *OECreateInChI* will no longer crash and instead throw a warning when the InChI library returns an empty string.
- *OEConfBase.GetTitle* will be preferred over the *OEMolBase.GetTitle* when writing multi-conformer *OEMCMolBase* molecules out to file formats that support titles. Previously, users were required to set the *OEMCMolBase* title to an empty string, `mol.SetTitle("")`, to achieve this behavior. This code will continue to work, but is no longer necessary to get the more specific conformer titles to be written out. `OEB` files will continue to write out both titles.
- *OEConstCoords* and *OEMutableCoords* will no longer crash when given an empty molecule.

Minor bug fixes

- *OE3DToAtomStereo*, for the perception of atom stereo from 3D coordinates, depends on the following properties: *OEPerceived.RingAtomsAndBonds*, *OEPerceived.Aromaticity*, and *OEPerceived.Chiral*. If any of these properties are un-set when calling *OE3DToAtomStereo*, they will be automatically reperceived before calculating the atom stereo properties.
- *OE3DToBondStereo*, for the perception of bond stereo from 3D coordinates, depends on the following properties: *OEPerceived.RingAtomsAndBonds*, *OEPerceived.Aromaticity*, and *OEPerceived.Chiral*. If any of these properties are un-set when calling *OE3DToBondStereo*, they will be automatically reperceived before calculating the bond stereo properties.
- *OEOFlavor.ISM.ExtBonds* will no longer be used when generating SMILES with *OEMolToSmiles*. This will cause *OEMolToSmiles* to generate the same canonical isomeric SMILES as when writing into an *OEFormat.ISM* file format.
- `OEChem::OEMCMolType::Torsion` and `OEChem::OEMCMolType::Rotation` constants have been removed.
- *OEInitHandler* called on the options specified by *OEBDefaultOpts* will now initialize the handler to the exact same set of handlers as *OEInitDefaultHandler*. The noticeable difference between the two was that *OEInitHandler* would inadvertently turn on writing out the *OEBondBase.GetIntType* field, resulting in larger file sizes. *OEBondBase.GetIntType* was also turned on for the *OEBRotCompressOpts* options as well. *OEBondBase.GetIntType* will now only be turned on by the *OEBAtomIntTypeOpts* options.
- *OEMolBaseType.OEMiniMol* will now copy coordinates of other molecule implementations during copy construction.
- *OEMCMolBase* read through *OEConfTestBase* from non-SDF file formats should have a slightly smaller memory footprint. The `MDLParity` flag was always being stored in generic data, regardless of whether the molecule had the property or not.
- *OEWriteMolecule* will no longer change the active conformer of an *OEMCMolBase* while writing to file formats that do not support multi-conformer molecules like SDF and MOL2.

- *OEWriteConstMolecule* will no longer ignore the flavors specified by the *oemolostream* for the *OEFormat.XYZ* format.
- Warning message about an invalid format used with *OEReadMolecule* was previously saying there was a problem with *OEWriteMolecule*. This was incorrect, and the message now states "OEReadMolecule" instead.
- Molecules read through *OEMolDatabase.GetMolecule* will no longer have their indices written to the OEB file format. This was just causing file size bloat and possibly confusing behavior with *OEHasMolDBIdx* when the molecule was written and then read again from OEB through an *oemolistream*.
- *OEMolDatabase.GetTitle* fixed for PDB files that include data beyond the 72nd column.
- Some redundant information is removed from OEB files written from *OEMCMolBase* objects. This should make OEB files the exact same size when round-tripping molecules now instead of becoming slightly larger.
- *OEFormat.SDF* and *OEFormat.MDL* file formats no longer support the long deprecated CPSS reaction fields. More deprecated atom and bond fields in the V2000 molfile format continue to be exported but will be removed entirely in future versions.
- *OEMDLSetParity* added to allow explicitly setting the chiral flag status. This was added for symmetry with *OEMDLGetParity* and to allow explicit control of the exported chiral flag state by the user.
- SMILES containing explicit hydrogens in stereo configurations were being non-optimally depicted. SMILES parsing will now set the reaction role (*OEAtomBase.SetRxnRole*) of stereo explicit hydrogens.
- *OEMCMolBase.NewConf* will now return a `NULL` *OEConfBase* pointer instead of crashing when passed a `NULL` coordinates pointer.

Documentation fixes

- *OEMolBaseType* is now documented.
- *OEOFlavor.XYZ* is now documented.
- All OpenEye code examples will now prefer to use *OEMolToSmiles* and *OESmilesToMol* instead of the `OECreate*SmiString` and `OEParseSmiles` functions.

5.66.2 OEBio 2.0.2

- Minor internal improvements.

5.66.3 OESystem 2.0.2

New features

- Small performance improvements when writing generic data to OEB files.
- *OEGetHostEndian* and *OEGetHostEndian* added to make template programming generic binary readers and writers easier.
- *OEGeom3DAllZero* added to determine whether an array of floating point numbers is all zero.
- Many `OEMath::OEGeom` functions can now automatically up-cast *OEHalfFloat* to float during their operation making working with *OEHalfFloat* more seamless and requiring less temporary arrays.
- *OEGeom3DReflect* added to reflect coordinates around any given axis.
- *OECopyArray* should now be significantly faster for `long double` data arrays.

Documentation fixes

- *OEBitVector.ToHexString* more thoroughly explained.
- *OEBitVector* documentation table was malformed.

5.66.4 OEPlatform 2.0.2

New features

- *OEThread.Start* now returns a boolean value to indicate whether thread creation has failed.
- *OEHalfFloat* can now interconvert with `double` and `long double` more easily.

Minor bug fixes

- *OEFileRandomName* will now throw a warning that it is deprecated and will be removed in a future release. It is very dangerous and prone to race conditions.
- *OEMutex* data members are no longer exposed as `public` access on Windows.

Documentation fixes

- *OEThread* is now documented.

5.66.5 OEGrid 1.4.7

- Minor internal improvements.

5.67 Version 2.0.1

5.67.1 OEChem 2.0.1

New features

- *OEMolBase.SetCoords* will now reset the atom and bond stereo perception flags, *OEPerceived.AtomStereo* and *OEPerceived.BondStereo* respectively. This can force reperception of the stereo information from the new coordinates by other functions that may need it. For example, a later call to *OEWriteMolecule* will now automatically call *OE3DToAtomStereo*.
- *OECopyMol* function added to make a copy of a molecule and return the atom or bond associations between the source and destination molecules.
- *OEMolBase.ClearBase* method added to allow clearing generic data on the *OEMolBase* without destroying the molecule itself.
- *OEUniMolecularRxn.SetValidateKekule* method added to allow skipping the Kekule validation step after the reaction is applied. Note, this allows *OEUniMolecularRxn* to return radicals, leaving clean up to the user.
- *OESuppressHydrogens* can now suppress hydrogens at specific heavy atom sites.
- *OENthAtom* predicate's 2nd argument now defaults to `0` to be easier to use.
- *OEGetMolDatabaseIdxFileName* added to generate index file names from database file names using the same rules as *OECreateMolDatabaseIdx* and *Save*.

Major bug fixes

- *OEMMFFAtomType* will no longer crash whenever an unbonded sulfur with a single implicit hydrogen is encountered.
- *OEAssignAromaticFlags* now invokes the *OEFindRingAtomsAndBonds* function, when necessary. If no ring atoms are detected in the molecule, the *OEAssignAromaticFlags* functions returns after setting all atoms and bonds of the molecule to be non-aromatic.
- The *OESubSearch* class now throws a warning in the following two cases:
  - If there are only explicit hydrogens in the pattern, but the target has only implicit hydrogens.
  - If there are only implicit hydrogens in the pattern, but the target has only explicit hydrogens.

  These warnings are necessary in order to avoid substructure match failures that are the result of mismatched hydrogen representation.
- The *OEGetFileType* function now returns *OEFormat.UNDEFINED* for file extensions that contain extra characters after the known file extension. For example, `ismama` used to inappropriately return *OEFormat.ISM*. The only exception is if the extra characters are preceded by ".". Therefore, the following strings are still considered valid: `sdf.gz`, `sdf.bin`, and `sdf.zip`.
- V3000 entries with a broken atom or bond block will no longer crash, the reader will now throw a warning and skip the molecule.
- *OEMDLPerceiveBondStereo* function will no longer clear wavy bonds, *OEBondStereo.Wavy*, since this property can not be re-perceived from other information. The MDL bond stereo can be removed by calling the new *OEMDLClearBondStereo* function.
- *OEWriteMDLFile* no longer has a subtle race condition that could lead to incorrect time stamps being written in the header. The race condition was caused by the use of the standard C library `localtime` function.
- *OEChem::oemolithread::open* will no longer crash when given an empty `.csv` file.
- *OEReadMDLReactionQueryFile* previously incorrectly required "M END" lines between reactant and product blocks. These lines are no longer required and will be ignored if they are encountered. This change allows reaction query files to be read that are generated by ChemDraw and the Accelrys Enterprise Platform.

Minor bug fixes

- *OEMolDatabase.GetTitle* will no longer return a corrupted title for multi-molecule `.mol2` files.
- *OEGenerate2DCoordinates* will now handle a molecule as a reaction if any of its atoms have a reaction role other than *OERxnRole.None*. Previously, only molecules that returned `true` from *OEMolBase.IsRxn* would be handled as reactions.
- **OEChem 2.0.0**, the previous release, introduced automatic 2D coordinate generation when writing to the following file formats: *OEFormat.SDF*, *OEFormat.MDL*, and *OEFormat.CDX*. However, coordinate generation will only occur if *OEMolBase.GetDimension* does not return 2 or 3, indicating that the molecule already contains coordinates. *OEWriteMolecule* will now throw a warning message if coordinates are detected on the molecule and *OEMolBase.SetDimension* was not appropriately called. Users should either call *OEMolBase.SetDimension* to the appropriate dimensionality of the molecule, or clear the molecule's coordinates.

  > **Note**
  >
  > `.cdx` files will still automatically generate 2D coordinates from 3D molecules since `.cdx` is a 2D molecule format only.

- *OEAssignAromaticFlags* used to only set atoms to non-aromatic if no ring bonds were found. Bond aromaticity will now be cleared as well if no ring atoms are found.
- *OECreateMolDatabaseIdx* will no longer crash whenever called inside a directory without write permissions.
- *OEChem::oemolostream::seekp* removed since it did not do anything.
- *OESmartsLexReplace* argument is now `const` since the function does not need to alter the vector.
- Removed unbounded stack allocations.
- The partial support for CPSS-style rxnfiles has been entirely removed as this format is documented as obsolete in the CTfile format specification.
- *OEAddExplicitHydrogens* no longer has the rare chance of using uninitialized memory when sprouting hydrogens in 3D.
- *OEReadMDLQueryFile* will now interpret `![N,O,S]` as `![N,O,S,H]` to be consistent with MDL matching behavior.
- *OEOmegaConfTest.CompareMols*, *OEIsomericConfTest.CompareMols*, *OEAbsoluteConfTest.CompareMols* will all now return `true` if the only difference between the bonds of the two molecules is whether the `Bgn` and `End` bonds have been swapped.
- The *OETorsion* parameters to *OEMCMolBase.NewConf* are now properly wrapped.

Documentation fixes

- All examples should now use *OESmilesToMol* instead of *OEParseSmiles*. *OESmilesToMol* is the preferred safer method for parsing SMILES.
- Simplified the documentation for the basic molecule class hierarchy. Previously, documentation was spread between the base class and the concrete classes. The base class sections should now be the canonical place to look for specific method documentation.
  - For *OEGraphMol* methods, see the *OEMolBase* section
  - For *OEMol* methods, see the *OEMCMolBase* section
  - For *OEQMol* methods, see the *OEMolBase* section
- Added documentation for *OEConfBase* and *OEMCMolBase* classes that are instantiations of `OEChem::OEConfBaseT` and `OEChem::OEMCMolBaseT` template classes, respectively.
- Added missing documentation for the *OEConfBase* and *OEMCMolBase* classes.
- Generated new images for the *Explicit Hydrogens* section of the *Substructure Search with MDL Queries* chapter
- Memory leak fixed in `NewConfFromCoords` documentation example.
- Corrected a link to the CTfile format document from Accelrys, Inc.
- *OEProperty* namespace formatting fixed in PDF files to actually be readable.
- *OEOmegaConfTest* added to the conformer test section for *Input and Output*.
- C# code snippets added to the *Molecular Database Handling* chapter.

5.67.2 OEBio 2.0.1

Major bug fixes

- *OESetRotamer* can properly set rotamers with angles of 0.0. Previously the angle was silently left unchanged.

5.67.3 OESystem 2.0.1

New features

- *OEInterface* can now report all the OpenEye toolkit libraries being used by specifying `--help versions` on the command line.
- *OEParseCommandLine* now accepts an argument for specifying the program's version number. This version number will be printed whenever `--version` is found on the command line.
- *OEMemPoolMode.Spinlocked* memory pool mode added to provide a more efficient mutual exclusion mechanism than mutexes for the small object caches.
- Added the following functions to provide predefined sets of colors:
  - *OEGetColors*
  - *OEGetContrastColors*
  - *OEGetDeepColors*
  - *OEGetLightColors*
  - *OEGetVividColors*
- Added the following colors:
  - *OESystem.OEBrass*
  - *OESystem.OECopper*
  - *OESystem.OEGold* (the original color was renamed to *OESystem.OEMandarin*)
  - *OESystem.OESilver*
  - *OESystem.OEPewter*

Minor bug fixes

- Removed unbounded stack allocations.
- *SetDate* no longer has a subtle race condition due to the use of the standard C library `localtime` function.

Documentation fixes

- Added missing documentation for the following colors:
  - *OESystem.OEGreenBlue*
  - *OESystem.OEGreenTint*

5.67.4 OEPlatform 2.0.1

New features

Major bug fixes

- *OEGetPlatformInfo* will now return Windows versions for `Windows Vista` through `Windows 8.1`.
- *OEGetProcessorInfo* would previously only show Pentium III for newer chips on Windows. The function will now interrogate the chip using the `cpuid` instruction.

Minor bug fixes

- *OEAddLicenseFromHttp* will now accept HTTP responses with a content-type of `text/html; charset=utf-8`. Previously, only `text/plain` was accepted.

Documentation fixes

- *OEAddLicenseFromHttp* now includes example code in the documentation for its use.

5.67.5 OEGrid 1.4.6

New Features

- *OEMakeMolecularGaussianGrid* now has an overload to allow a grid with existing dimensions, mid-point, and spacing to be filled. Previously, the grid size would always be recreated based upon the size of the molecule.

Major bug fixes

- Better support for output of non crystallographic grids to CCP4 file. Now the CCP4 ARMS value is set to 1.0 and the CCP4 MEAN is set to 0.0 to prevent scaling and offsetting in CCP4 map visualizers. Additionally, the spacegroup is now output as 146 (H3) as opposed to the nonstandard 1146 (R3) to facilitate viewing in MOE. Please note that values appearing outside of the specified unit cell will be incorrect.

Documentation fixes

- Adding missing *OEMTZMapType* namespace to the documentation.
- OEChem release notes for 2.0.0 and 1.9.3 had the OEGrid version slightly wrong. The OEGrid version numbers have been corrected to 1.4.5 and 1.4.4 respectively.

5.68 Version 2.0.0

5.68.1 OEChem 2.0.0

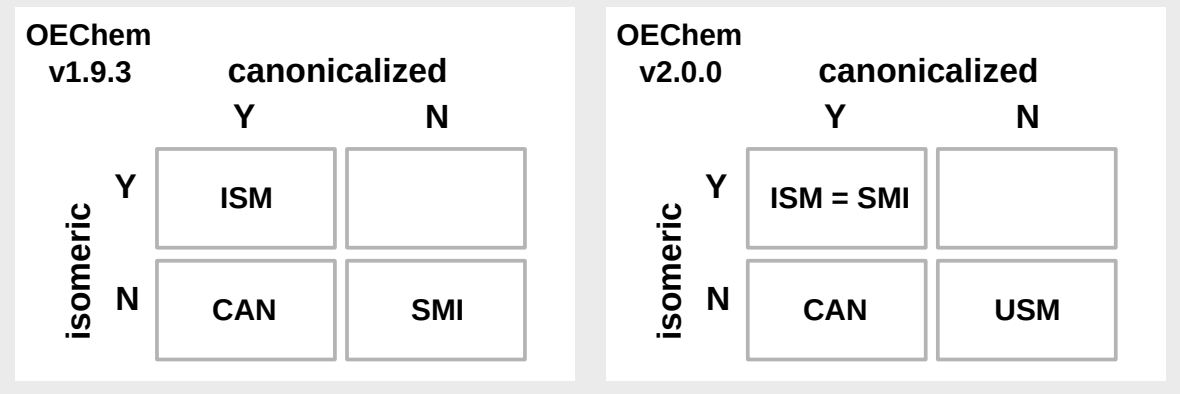
New features

- *OEMolDatabase* class added for providing fast random read-only access to all the file formats **OEChem** supports. Please see the new *Molecular Database Handling* chapter of the documentation for a more thorough description.

> **Warning**
>
> The meaning of the `.smi` file extension has changed in OEChem 2.0. The `.smi` file extension will now retain stereochemistry information. This could cause problems for systems that relied upon `.smi` to strip away stereo chemistry.

- The following changes have been made to how OEChem defines the various flavors of SMILES (see images demonstrating these changes in *Table: SMILES File Formats*):
  - The `.smi`, *OEFormat.SMI*, file format outputs Canonical isomeric SMILES.
  - *OEFormat.USM* file format added (with the `.usm` file extension) that allows the generation of non-canonical, non-isomeric SMILES, i.e., the OEChem 1.x definition of the `.smi` file extension.
  - *OEGetFormatString* returns more descriptive values for the various flavors of SMILES regarding canonicalization and stereo information.
  - The default file format for molecule streams is now *OEFormat.SMI* and is identical to the *OEFormat.ISM* format.

Table 9: **SMILES File Formats**



- *OEFormat.CSV* file format added to **OEChem** for round-tripping molecules and SD data to other software packages that support the comma-separate-value, `.csv`, file format. **OEChem** supports the `.csv` file format that has largely become ubiquitous and is now standardized by RFC 4180. The following APIs were added for handling CSV files in **OEChem**:
  - *OEReadCSVFile*
  - *OEWriteCSVFile*
  - *OECreateCSVHeader*
  - *OECSVHeader*
  - *oemolstreambase.GetCSVHeader*
  - *oemolstreambase.ClearCSVHeader*
- Added automatic 2D coordinate generation to the following file formats when invoking the *OEWriteMolecule* high-level molecule writer:
  - In case of the *OEFormat.MDL* and the *OEFormat.SDF* file formats, 2D coordinates are generated if the molecule has no coordinates *i.e.* existing 2D or 3D coordinates will be left intact.
  - In case of the *OEFormat.CDX* file format, 2D coordinates are generated if the molecule has no 2D coordinates.

  This default behavior can be turned off by using the *OEOFlavor.MDL.Add2D*, *OEOFlavor.SDF.Add2D* and *OEOFlavor.CDX.Add2D* flags, respectively.
- *OEGetFormatExtension* will now return the most common file extension as the first element in the comma separated list for the *OEFormat.PDB*, *OEFormat.SDF*, and *OEFormat.RDF* file formats.
- *OEGenerate2DCoordinates* added to assign 2D coordinates to the given molecule.
- *OEMatchBase.IsValid* and *operator bool* methods added.
- *OERoleSet* abstract base class added as a mixin class to allow a class to contain a set of *OERole* objects for classification purposes. This is similar to how the *OEBase* class provides associative data container behavior as "generic data". The following classes already derive from *OERoleSet*:
  - *OEMatchBase*
  - *OEAtomBondSet*

  More classes across the toolkits may be added in the future as dictated by needs.

- *oemolstreambase.GetFileName* method added to all molecule streams to return the file name used to open the stream, if a file name was used.
- *OEMatchBase.IsValid* method added for determining whether the match contains any atoms or bonds.
- *OEAssignZap7Radii* function added to assign radii from the ZAP7 set.
- *OEParseSmilesOptions* class added for adding more complex SMILES parsing options for the *OEParseSmiles* function. This added the ability to make *OEParseSmiles* quiet with regard to parsing failures through the *OEParseSmilesOptions.SetQuiet* method.

Major bug fixes

- The *OEFormat.MDL* V3000 file format was unable to read multiple molecules from the same file if they were all in the V3000 format. Note, this did not affect the *OEFormat.SDF* files containing V3000 as those files contain '$ $ $ $' to delimit separate molecule records.
- `OEChem::OEReadHeader` will no longer corrupt the stack by arbitrarily zeroing out bytes on the stack whenever no *OEHeader* record is found in the *OEFormat.OEB* file.
- *OEMatch.AddPair* will no longer crash if *OEMatch.Clear* was previously called.
- *oemolthreadbase.PutMol* will now destroy the pointer passed to it if the underlying buffer returns `false`, e.g., the buffer has already been closed.
- *OEAddMols*, *OESuppressHydrogens*, and *OEPerceiveSymmetry* will no longer cause a stack overflow and crash when the molecule has a large number of atoms.
- A subtle memory and performance bug was fixed for the *OEMol*, *OEGraphMol*, and *OEQMol* copy constructors. Every time a copy was made an extra layer of proxy object was created, causing memory consumption to increase, as well as making any subsequent use of that object slower.

Minor bug fixes

- Removed unimplemented `OEChem::OEFormat::TDT` file format.
- Removed `OEChem::OEFormat::BIN` format after being deprecated for 10 years.
- *OEMolToSmiles* no longer outputs the title of the molecule.
- If *OE3DToAtomStereo* or *OE3DToBondStereo* throws a warning message during a call to *OEMolToSmiles*, the warning message will no longer erroneously say it is during a call to `OEWriteMolecule`.
- *OESetComment* and *OEGetComment* have have been slightly optimized for speed. There is also a larger optimization for memory and file space in the `.oeb` format. Previously, setting the comment to an empty string would write superfluous data to the `.oeb` file.
- *OECopySDData* and *OECopyPDBData* will no longer increase the memory consumption of the destination molecule whenever the source molecule does not contain any data. This was causing *OEFormat.PDB* files read into a *OEMCMolBase* to use more memory than necessary.
- *OEFormat.MOL2* parser will no longer create bonds with zero-order from "dummy bonds". Zero-order bonds cause problems for many **OEChem** algorithms like *OEKekulize*.
- *OEFormat.MOL2* parser will now properly ignore lines between molecule records that start with the pound sign, "#". Previously, these lines would cause **OEChem** to spew a lot of warnings and cause the parser to fail.

Documentation fixes

- The chapter about InChI failures has been removed since most of the InChI failures were fixed in the last release, **OEChem** 1.9.3 in 2013.Oct.
- Release notes section re-organized to make the current release more prominent.

5.68.2 OESystem 2.0.0

New features

- Added support for handling the comma-separated-value, CSV, format specified by RFC 4180 with the following two free functions:
  - *OEStringCSVJoin* for creating a CSV record
  - *OEStringCSVTokenize* for parsing a CSV record

  The following low-level functions were added support CSV handling but can be ignored by most users:
  - *OEStringCSVQuote* added
  - *OEStringCollapseQuotes* added
  - *OEStringStripQuotes* added
- *OEHalfFloat* class added for storing floating point data as a 16-bit representation as specified by the IEEE 754-2008 standard to save on memory consumption and bandwidth.
- The performance of reading plain-old-data attached to *OEBase* objects from `.oeb` files has been improved by about %10.
- *OEErrorLevelToString* free function added.
- *OEBFPosTEndian* added for allowing 64-bit integers to be round-tripped to binary formats regardless of machine endian-ness.

Major bug fixes

- *OEThrow* mutex handling has been migrated from *OEErrorHandler* down a level into `OESystem::OEErrorHandlerStreamImpl`. This fixes the following issues:
  - The mutex can now be properly released during a process exit like *OEErrorHandler.Fatal*.
  - A deadlock will no longer occur if the implementation of *OEErrorHandlerImplBase.Msg* needs to throw a message itself.
  - Allows alternative faster and more scalable implementations of *OEErrorHandlerImplBase* to be created and used with *OEThrow*.

Minor bug fixes

- *OEBitVectorNumWords* and *OEBitVectorNumBytes* now take and return `size_t`.
- `OESystem::OEBinaryTagMaxLength` constant added and set to 1024, the maximum length of an explicit string tag in the `.oeb` file format.
- *OEStringTokenizeQuoted* will no longer treat a quote as the end quote of a field if it is escaped by another quote. *Table: OEStringTokenizeQuoted Change* demonstrates the change to support proper CSV parsing.

Table 10: **OEStringTokenizeQuoted Change**

OEChem 1.x	“foo””,bar”,blah	`"foo""``|``bar"``|``blah`
OEChem 2.0+	“foo””,bar”,blah	`"foo"",bar"``|``blah`

- The size of *OEBitVector* object has been increased from 12 bytes to 16 bytes on 64-bit machines. The size is still 8 bytes on a 32-bit machine.
- The *OEErrorHandlerImplBase* copy constructor is now usable in Python, Java, and C#.

Documentation fixes

- *IsTrue* and *IsFalse* are deprecated and will be removed in `OEChem 3.0`, please convert to using *OEIsTrue* and *OEIsFalse*.
- *OEUnaryTrue* and *OEUnaryFalse* documented as acceptable synonyms of *OEIsTrue* and *OEIsFalse*.

5.68.3 OEPlatform 2.0.0

New features

- *OEAddLicenseFromHttp* added to license the current process with an OpenEye license file retrieved from a valid URL. The only supported URL schemes are HTTP and HTTPS. A reference implementation of a license server can be found at: (https://github.com/oess/HttpLicenseServer).
- *OEAddLicenseData* function added to parse a string as if it is an OpenEye license file and then license the current process with it.

Major bug fixes

- *OEPlatform::oeistream::size* was incorrectly being truncated to 32-bits on 32-bit machines. This resulted in incorrect file sizes being reported for files over 4 gigabytes in size on 32-bit machines.
- *OEPlatform::oeifstream::tell* would return incorrect values if called after *OEPlatform::oeifstream::size* and after some bytes were already read from the stream.

Minor bug fixes

- `OEPlatform::OEMallocaPtr::GetPtr` added to allow the object to be explicitly converted to a pointer type.
- *OEMutex* and *OETryMutex* destructors will now destroy themselves on `pthread` based systems. Destroying mutexes is optional according to the `pthread` standard, but helpful in debugging possible deadlocks.
- *OEMutex* and *OETryMutex* no longer use `gthreads`, instead using `pthreads`, allowing for integration with non-GCC systems like libc++ on OSX.

5.68.4 OEGrid 1.4.5

Major bug fixes

- *OEGridFileType.Ascii* file writer will no longer sometimes corrupt the stack and crash.
- When reading a CCP4 file, the standard deviation stored in the CCP4 header is now used to normalize the file.
- OENormalizeGrid now properly normalizes by sigma (not variance)
- Writing CCP4 maps now uses the original map statistics when possible.
- Rotated skew grids attached as generic data are now round-trippable when saved to OEB
- Fixed a crash when interpolating grids where the rotation matrix inverts the target grids bounding box
- MTZ files with more than 18 columns are now read properly
- Fixed a memory leak in *OESequenceAlignment*.
- The constructor for the predicate *OEHasResidueNumber* now takes an int rather than an unsigned int because residue numbers can be negative.

5.69 Version 1.9.3

5.69.1 OEChem 1.9.3

New features

- Added the following atom and bond predicates:
  - *OEAtomIsInChain*
  - *OEBondIsInChain*
  - *OEIsHetero*
  - *OEIsMetal*
  - *OEHasHvyDegree*
  - *OEHasFormalCharge*

Major bug fixes

- `OECreateInChI` now passes the following information to the InChI library when calling the `OECreateInChI` function:
  - atom coordinates
  - wedge/hash bond property for single bonds
  - "either" bond property read from MDL file formats for double bonds

  These changes have reduced the number of failure cases to 10 when reading the MDDR database in SDF format and testing the InChI strings generated by `OECreateInChI` against the strings generated by the stand-alone InChI application.

Minor bug fixes

- The `OEIsAtomHybridization.operator()` method now checks whether the hybridization was perceived and calculates the hybridization of the atom on the fly by calling `OEGetHybridization` if necessary.
- The previously release of **OEChem**, 1.9.2, started considering different isotopes of hydrogen for chirality perception. This incorrectly perceived the following molecules as chiral: "C[2H]", "CC[2H]" and "C([2H])[3H]". Atoms with one or two isotope hydrogen neighbors and at least two implicit hydrogens will no longer be considered chiral.
- When reading MacroModel molecule files (.mmod, .mmd, and .dat), atom serial numbers and HETATM designations were not being set, and subsequently saving the molecules in PDB format resulted in poorly formed files. This has been corrected so that conversion from MacroModel to PDB files preserves residue information properly.
- `OESmilesToMol` and `OEInChIToMol` will now call `OEMolBase.Clear` before parsing the string into the molecule.

Documentation fixes

- The `OEMMFF*` functions are now properly documented.

5.69.2 OESystem 1.9.3

Major bug fixes

- `OESystem::OEErrorHandlerStream::Msg` now calls `exit(EXIT_FAILURE)` in case of *OEErrorLevel.Fatal* level and calls `exit(EXIT_SUCCESS)` in case of *OEErrorLevel.Usage*.
- *OEInterface* will no longer strip trailing zero characters from parameters that are non-floating point types. For example, "-prefix 1.10" used to return "1.1". This bug was introduced in OEChem 1.9.0.

5.69.3 OEPlatform 1.9.3

- *oeistream.gettoken* will no longer improperly write a NULL character at `buffer[max]` location in memory in rare circumstances when the token is the exact same size as the buffer.

5.69.4 OEGrid 1.4.4

- Fixed a rare crash when interpolating between grids. This affects many API points, including GridBabel, whenever converting a grid to another grid type.

5.70 Version 1.9.2

5.70.1 OEChem 1.9.2

New features

- OEChem now supports the creation of InChI. OEChem produces the same InChI as the InChI library provided application for 99.77% of MDDR in the SDF format. The differences fall into two categories:
  - Corrupted molecules where the tools have to make arbitrary decisions about how to correctly interpret the data.
  - Bond stereo being perceived by the InChI application where bond stereo does not actually exist. OEChem does not assign bond stereo and generates what appears to be a more "correct" InChI for these cases.

  The inclusion of InChI support added the following:
  - *OECreateInChI*
  - *OECreateInChIKey*
  - *OEFormat.INCHI*
  - *OEFormat.INCHIKEY*
  - *OEOFlavor.INCHI*
  - *OEOFlavor.INCHIKEY*
  - *OEInChIOptions*

  In addition, *oemolostream* will automatically write InChI when using the `.inchi` file extension.
- *OEPrepareSearch* added that perceives atom and bond properties of a molecule that are necessary to successfully execute a given substructure search. It is now highly recommended to run this function on a molecule before passing the molecule to any of OEChem's graph matching algorithms.
- Changed the default input and output molecule file formats for molstreams and molthreads from *OEFormat.SMI* to *OEFormat.ISM*.
- *OEWriteMDLFile* writes MDL enhanced stereo groups.

- *OEWriteMDLFile* will now automatically write a molecule in the V3000 file format if it has more than 999 atoms or if it has any MDL enhanced stereo group regardless of the given file format flavor.
- *OESetMCSExhaustiveSearchTruncationLimit* and *OEGetMCSExhaustiveSearchTruncationLimit* functions added that control how thoroughly the search is performed by the *exhaustive* MCS search.
- Added the following standard SYBYL atom types: *OETriposType.Cu* *OETriposType.Cooh*, *OETriposType.Croh*, *OETriposType.Crth*, *OETriposType.Fe*, *OETriposType.Mg*, *OETriposType.Mn*, *OETriposType.Mo*, *OETriposType.Se*, *OETriposType.Sn*, and *OETriposType.Zn*. Atoms for which no SYBYL atom type exists are still internally handled with the *OETriposType.Du* type, however when writing these atoms into a `MOL2` file their atomic symbol is written to the SYBYL atom type column instead of a string "Du".
- Added *OEGetDimensionFromCoords* function that returns the dimension of the molecule from its coordinates.
- Residue information is now retained when reading files with MacroModel formats (.mmod, .mmd, and .dat).

Major bug fixes

- It used to be possible to perform a substructure search for stereochemistry or hybridization on a molecule that did not yet have those properties perceived. This led to very subtle to detect problems where molecules would not match that were expected to match.

  To combat this, the following *OESubSearch* methods now throw warnings in case some property that is necessary to successfully execute the substructure search has not been perceived yet:

  - *OESubSearch.Match*
  - *OESubSearch.SingleMatch*
  - *OESubSearch.AtomMatch*

  These warnings can be suppressed by calling the *OEPrepareSearch* function on the molecule first.
- Utilizing the *OEPerceived.AtomStereo* and *OEPerceived.BondStereo* flag to mark whether the atom and bond stereo information is perceived in a molecule.
- It is now thread-safe to copy the same *OEMol* from multiple threads at the same time. This was caused by some *OEMCMolBase* methods not actually being const. The following methods were marked as const, but were not actually const, leading to subtle race conditions:
  - *OEConfBase.GetTitle*
  - *OEMCMolBase.GetActive*
- *OEMCMolBase.GetMCMolTitle* method added to provide a way to differentiate between a top-level *OEMCMolBase* title and a conformer title.
- *OEMCSSearch.AddConstraint* now requires that the constraint is satisfied in all resulting matches. Previously, the documentation stated the following, "Constraints are considered satisfied in subgraphs which do not contain any constrained atoms or bonds in either the pattern or target molecules." This is no longer the intended behavior.
- *OEMCSSearch.AddConstraint* now returns `false` if passed the same pattern atom or bond twice. The user must now use *OEMCSSearch.ClearConstraints* in order to rebuild a set of constraints.

Minor bug fixes

- *OE3DToBondStereo* function returns false if the molecule dimension is 0, since no cis or trans bonds can be perceived if there are no coordinates.
- Fixed a problem when the max matches value of the *OESubSearch* was reset to the default value `1024` every time the search was initialized by the *OESubSearch.Init* method.
- Adjusted the dimension of a molecule to 0 if it is read from either a `MDL` or an `SDF` file with no coordinates.

- *OE3DToAtomStereo* function throws a warning and returns `false` if any chiral atom of the molecule is not tetrahedral, i.e., if the atom stereo can not be determined from the 3D coordinates, usually because it is flat.
- The following functions and methods throw additional warnings in cases where either the *OE3DToAtomStereo* or the *OE3DToBondStereo* functions encounter any problems when assigning stereo from coordinates:
  - *OEPrepareSearch* functions
  - *OEWriteMolecule* function when writing the following formats: *OEFormat.MDL*, *OEFormat.SDF* *OEFormat.ISM*, *OEFormat.INCHI* and *OEFormat.INCHIKEY*
  - *OEAbsoluteConfTest.CompareMols* method
  - *OEIsomericConfTest.CompareMols* method
  - *OEAbsCanonicalConfTest.CompareMols* method
  - *OEOmegaConfTest.CompareMols* method
  - *OEDefaultConfTest.CompareMols* method
- *OESubsetMol* now produces correct results when the `RGroup` flag was `true` and the molecule contained *OEGroupBase* objects.
- *OEPerceiveChiral* function considers hydrogens with different mass as being different *i.e.* it will recognize the C atom in a molecule "[3H]C(N)F" as a chiral atom, since it has four different neighbors: N, F, H and 3H.
- The first argument of the following methods is not const any longer:
  - *OEAbsoluteConfTest.CompareMols* method
  - *OEIsomericConfTest.CompareMols* method
  - *OEAbsCanonicalConfTest.CompareMols* method
  - *OEOmegaConfTest.CompareMols* method
  - *OEDefaultConfTest.CompareMols* method
- Improved the performance of the following methods:
  - *OEAbsoluteConfTest.CompareMols* method
  - *OEIsomericConfTest.CompareMols* method
  - *OEOmegaConfTest.CompareMols* method
- *OEAtomBondSet.AddAtom* and *OEAtomBondSet.AddBond* will now return false when the trying to mix atoms from different molecules.
- *OEAtomBondSet.HasAtom* and *OEAtomBondSet.HasBond* added for testing for membership.
- The following functors are now properly wrapped in Python, Java, and C#:
  - *OEIsTrueAtom*
  - *OEIsFalseAtom*
  - *OEIsTrueBond*
  - *OEIsFalseBond*
  - *OEIsTrueConf*
  - *OEIsFalseConf*
- The following functors are now deprecated:
  - *OEChem::IsTrueAtom*

- *OEChem::IsFalseAtom*
- *OEChem::IsTrueBond*
- *OEChem::IsFalseBond*
- *OEChem::IsTrueConf*
- *OEChem::IsFalseConf*

Documentation fixes

- The `OEAroModel` namespace is now accessible from the table of contents.

5.70.2 OESystem 1.9.2

New features

- Added the following non-linear color gradients:
  - *OEExponentColorGradient*
  - *OEExponentialColorGradient*
  - *OELogarithmicColorGradient*

  These classes along with the *OELinearColorGradient* class now derive from the *OEColorGradientBase* base class.

Minor bug fixes

- Fixed a possible link-time race condition with `OEGetTag` that could cause a crash if `OEGetTag` was used during link-time (library loading).
- *OEColor* could encounter a benign race condition during color clamping operations.

5.70.3 OEPlatform 1.9.2

Major bug fixes

- Calling *OEPlatform::oeofstream::append* after *OEPlatform::oeofstream::open* will no longer hang indefinitely once data is written to it.

Minor bug fixes

5.70.4 OEBio 1.9.2

- Minor internal improvements.

5.70.5 OEGrid 1.4.3

Minor bug fixes

- *OESkewGrid* now properly initializes its base class *OEGridAbstractBase* during construction.
- The `OEGridFileType.Ascii` format will now try harder to detect any incorrect file formatting issues and warn about them.

Documentation fixes

- The `ASCIIWriter` documentation example in the *Grid Input and Output* section was incorrect as it was writing out both grid indices and grid values.

5.71 Version 1.9.1

5.71.1 OEChem 1.9.1

New features

- *OEOmegaConfTest* has been created specifically for output from OMEGA. This conftest ignores invertible nitrogen stereochemistry that OMEGA freely adjusts in order to allow these conformers to be read into the same *OEMol*.
- *OELibraryGen.SetValidateKekule* added to set whether alternative Kekulé forms are enumerated until a valid form is identified before returning a product.
- The *oemolthreadbase.GetMol* method for *oemolithread* class will now return chunks of molecule files that can be directly written to files to enable faster tokenization of a molecule file. Before this release, this method would often not return whitespace that was necessary to delineate boundaries between molecules.
- *oemolthreadbase.PutMol* and *oemolthreadbase.GetMol* support added to the wrapped languages: Python, Java, and C#.
- *OETrans* and its sibling classes for high-level 3D manipulation have been added to C#. These include *OEQuaternion*, *OEEuler*, *OERotMatrix* and *OETranslation*.

Major bug fixes

- *OEPerceiveResidues* had a bug introduced in the 2012.Feb release that could cause crashes or strange data to be written to the *OEAtomBase.GetName* field whenever *OEAtomBase.GetAtomicNum* returned `0`. Atoms with an atomic number of `0` will now be given the name of `` UNK`` (note, the leading space to match PDB style names).
- *OEAssignAromaticFlags* no longer takes an `OEAroModel`, which was a `unsigned short *` typedef, but takes an integer from the *OEAroModel* namespace instead. Global constants prepended with `OEAroModel*` are now deprecated. This works around a serious bug that would sometimes cause *OEAssignAromaticFlags* to not do anything to the molecule. This may break source-level backwards compatibility for strongly typed languages (C++, Java, and C#) if a global constant prepended with `OEAroModel*` was stored as a variable.

Code may have to be ported like the following:

```
SWIGTYPE_p_unsigned_char m = OEChem.OEAroModelOpenEye;
// change the previous line to the following
uint m = OEAroModel.OpenEye;

OEAssignAromaticFlags(mol, m);
```

Minor bug fixes

- *oemolithread* and *oemolothread* can no longer to be copy constructed. They are heavy weight objects representing real operating system threads and thus should not be copied.
- The MDL V3000 reader now throws a warning that the `ENDPTR` and `ATTACH` bond properties will be ignored.
- *OEMatch.AddPair* now checks whether the atom and bond pairs belong to the same molecule.
- *OELibraryGen* now allows an aromatic bond to become a triple bond when breaking a ring.

5.71.2 OESystem 1.9.1

New features

- Adding a default constructor for the *OELinearColorGradient* class that generates a color gradient with no color stop. The *OEColorGradientBase.IsEmpty* was added to be able to determine if the object was default constructed. Additionally, the *OEColorGradientBase.GetColorAt* method will always return the color *OEBlack* for a color gradient with no color stops, i.e., *OEColorGradientBase.IsEmpty* returns `true`.
- *OEStringTabsToSpaces* function added to expand tab characters to 4 space characters.
- *OEBitVector.FromHexString* performance has been significantly improved.

Minor bug fixes

- *OEStringTokenizeQuoted* now treats consecutive delimiters as blank fields.
- *OEErrorHandler*, and thus *OEThrow*, will no longer record the number of messages that did not actually get through due to the message not meeting the *OEErrorHandler.SetLevel* threshold.
- *OEParseCommandLine* will now call `exit(0)` if the internal call to *OECheckHelp* returns `true` that a `--help` parameter is found. That is, program execution with the `--help` parameter should not be considered abnormal program termination.
- *OEGeom3DRotVectorToTransform* return type changed from `void` to `bool` to be able to indicate cases when the function can fail.

Documentation fixes

- *OEBitVector.FromHexString* and *OESystem::OEBitVector::ToHexString* documentation has been corrected to explain their behavior with respect to bit vector lengths and the handling of non-hexadecimal digits in the translation process.
- *OECheckHelp* is now documented.

5.71.3 OEPlatform 1.9.1

New features

- *oeifstream.pread* method added to allow multiple threads to read from a single file.
- *oeiofstream* class added to support reading and writing to the same open file.
- *oeistream.read* method added to read the entire contents of the stream or throw `std::bad_alloc` whenever the system runs out of memory.
- *oemultiostream* has been added. More than one *oeostream* can be added to this class for simultaneous output.
- *OEFileTempDescriptor* added to safely create temporary files without race conditions. *OEFileRandomName* is now deprecated as it can lead to race conditions in the file names it generates.

Major bug fixes

- *OEPlatform::oeifstream::getline* would sometimes cause buffer overflows whenever `max` characters were actually in the line. The documentation was not very clear what this `max` parameter meant, it used to be the line size, not including the null terminator. Now it means the size of available memory in the `buffer` passed in, including the null terminator, and the documentation is now more clear on the subject.
- *OEPlatform::oeifstream::append* will now close the previous file before opening the next file. Before, calling the `append` method in a loop could lead to the process easily running out of a available file descriptors.
- The macros `OE32BIT` and `OE64BIT` were improperly set on 64-bit Windows.

Minor bug fixes

- *OEGetHostIdent* fixed to work on machines where `eth0` was not in use for network traffic.

5.71.4 OEBio 1.9.1

Minor bug fixes

- *OESequenceAlignment* methods *GetBackboneMatch* and *GetCAlphaMatch* now returns *OEMatchBase* instead of *OEMatch*.

Documentation fixes

- *OERotamer* class is now documented for working with frequently observed conformations of amino-acid sidechains .

5.71.5 OEGrid 1.4.2

Major bug fixes

- *OEReadGrid* no longer leaks memory for *CCP4* files.

5.72 Version 1.9.0

5.72.1 OEChem 1.9.0

New features

- Added support for reading the `V3000` file format for molecules, queries, and reactions. By default, whether to read the `V3000` format will be detected automatically from the header inside `.sdf`, `.mol`, and `.rxn` files.
- *OEGroupBase* class added as a container of atoms and bonds inside a *OEMolBase* to store MDL enhanced stereo information. See the *OEGroupType* namespace for the different types of enhanced stereo information that can be processed. Only the following subset of **OEChem** functions handle *OEGroupBase* information at this time:
  - *OEAddMols*
  - *OECount*
  - *OEReadMDLFile*
  - *OEReadMDLQueryFile*
  - *OEReadMDLReactionQueryFile*
  - *OESubsetMol*

  By virtue of the above low level reader functions populating *OEGroupBase* information, *OEReadMolecule* from a `V3000` file format will also populate *OEGroupBase* information on a molecule.
- Adding the following predicates for handling *OEGroupBase* objects on molecules.
  - *OEHasGroupType*
  - *OEIsMDLStereoGroup*
  - *OEHasAtomInGroup*
  - *OEHasBondInGroup*
- *OEAssignZap9Radii* has been updated to use the method published in the 2010 Journal of Computer-Aided Molecular Design by Nicholls, Wlodek, and Grant. The old behavior can be accessed by passing constants found in the *OEZap9Version* namespace.

- *oemolithread* now works with CDX files.
- *OEIFlavor.PDB.FORMALCHARGE* input flavor for PDB file reading added. This flavor is provided for advanced users who wish to read the formal charge from the file and do their own bond and hydrogen perception. It is not yet turned on by default.
- SD data tags and value in SDF files can now be up to 64 kilobytes in length, up from the previous limit of just 4 kilobytes.
- *OEMolecularFormula* extended to include hydrogen isotopes deuterium (D) and tritium (T) in the molecular formula output. Note, the default behavior has not changed, the user is required to pass an additional parameter to enable this feature.
- The Amber C-terminal capping residue *OEResidueIndex.NME* is now supported. Support for the Amber C-terminal capping residue *OEChem::OEResidueIndex::NMH* is incomplete.

Major bug fixes

- *OESubsetMol* will no longer crash non-deterministically when setting the RGroup parameter to true.
- *OEInterface* had a rare one time initialization thread safety bug when dealing with molecule parameters that has been fixed.
- *OEGetResidueName* will now handle *OEResidueIndex.GLH* appropriately.
- *OEMCSType.Approximate* search method will now terminate once the *OEMCSSearch.GetMaxMatches* limit has been reached. The previous behavior was to keep enumerating all possible matches, then choose the first matches up to the limit. This significantly speeds up *OEMCSType.Approximate* searches that results in a lot of matches, e.g., matching a molecule onto itself.

> **Note**
>
> It is highly recommended to call *OESuppressHydrogens* before trying to perform a self match on a molecule.

Minor bug fixes

- *OEResidueHydrogens* will now throw an error check when attempting to assign a PDB atom name when there are more than than 9 hydrogens attached to a single atom.

5.72.2 OESystem 1.9.0

New features

Minor bug fixes

- *OEInterface* help messages now only print a reasonable number of digits when printing floating point values.
- *OEReadCDXFile* and *OEReadMolecule* now preserve stereochemistry when loading ChemDraw CDX files. Additionally, some incorrect warning messages have been removed from the CDX file reader.

Documentation changes

- Figures added to the documentation of *OEMDLCorrectBondStereo*, *OEMDLPerceiveBondStereo*, and *OEMDLStereoFromBondStereo* to help explain the differences between those functions.

5.72.3 OEPlatform 1.9.0

Minor bug fixes

- *OEFile.AbsPath* fixed to not have a leading / if the directory path specified is empty.

5.72.4 OEBio 1.9.0

Major bug fixes

- The method *OEHierResidue.GetAtom* now correctly interprets the input *OEPDBAtomName* parameter rather than failing to find the requested atom.

5.72.5 OEGrid 1.4.1

- *OEReadMTZ* is now accessible from C#.

5.73 Version 1.8.0

5.73.1 OEChem 1.8.0

New features

- *OERMSD* and its many related overloaded functions have been optimized using recent algorithms from the literature. The algorithm now solves a quaternion-based characteristic polynomial (Theobald-2005). For typical cases, this new algorithm runs 3-5 times faster than the prior algorithm for calculating either an optimum RMSD with or without an overlay transformation.
- A new PDB input flavor, *OEIFlavor.PDB.SecStruct* has been created, to control perception of protein secondary structure from PDB files. This flavor is on by default when using *OEReadMolecule*. When this flavor is on, secondary structure is automatically perceived using *OEPerceiveSecondaryStructure*. Otherwise, secondary structure information is taken from HELIX and SHEET records in a PDB file.

Major bug fixes

- *OEAssignAromaticFlags* performance is significantly improved for molecules with a large number of rings, such as buckyballs. Since *OEAssignAromaticFlags* is called when reading from every file format **OEChem** supports this solves problems where **OEChem** would appear to hang indefinitely trying to read certain molecules. *OEAssignAromaticFlags* will now automatically adjust the maximum path considered for aromaticity perception based upon the following heuristics:
  - If a molecule has more than 15 rings, than the maximum path considered for aromaticity is 50.
  - If a molecule has more than 30 rings and is potentially a buckyball, than the maximum path considered for aromaticity is 30.

  In the case of buckyballs and other fused ring systems of 5 and 6 membered rings, this change has no effect other than to significantly improve the speed of the perception.
- Isomeric canonical smiles improved in the following ways:
  - Performance improvement to isomeric canonical smiles generation by not performing the more expensive isomeric canonical smiles generation on disconnected components that do not contain stereo information.
  - Very complex ring system corner case fixed that would cause isomeric canonical smiles generation to run for longer than was required leading to slightly different isomeric canonical smiles.

Minor bug fixes

- *OEBuildMDLQueryExpressions* fixed to properly evaluate expressions with zero charge as a valid atom property.
- *OEMDLStereoFromBondStereo* no longer throws a spurious platform dependent warning about '"Stereochemistry corrected on atom number # of #"' when the 2D atom coordinates around a tetrahedral center were all 90 degrees from each other.
- *OEParseSmiles* no longer crashes when passed a `NULL` pointer as a string.
- *oemolistream.GetReadFailureCt* now returns a non-garbage number. The internal counter was not always properly initialized to `0`, requiring the user to first call *oemolistream.ResetReadFailureCt* to get meaningful results.
- *OEAssignAromaticFlags* now resets all atom aromaticity flags to `false` if no rings are found in the molecule.
- *oemolistream.seek* and *oemolistream.tell* now works as expected for `MOL2` files. This was accomplished by making *OEReadMol2File* robust enough to not care whether `@<TRIPOS>MOLECULE` was encountered, and continue to read the rest of the molecule regardless of its existence. Additional checks are now done on subsequent lines to ensure the file is indeed in the `MOL2` format.
- *OEResidueHydrogens* has been fixed to remove a possible buffer overflow.
- *OEResidueHydrogens* now preserves assigned atom names for UNL residues.
- *OEAltLocationFactory* constructors now correctly processes the value of the constructor option *OEAltLocFactOption.TrustInputBonds* rather than always trusting input bonds.

5.73.2 OEBio 1.8.0

New features

- Several new functions have been added for the perception and handling of protein secondary structural elements:
  - *OEPerceiveSecondaryStructure*
  - *OEGetSecondaryStructureName*
  - *OEGetSecondaryStructureID*
  - *OEGetSecondaryStructureStrandID*
  - *OEGetSecondaryStructureType*
  - *OESecondaryStructurePacked*
  - *OEHasSecondaryStructure*

  Sheets, helices, and turns are perceived using the method of Kabsch & Sander **1983**, and are stored in atoms' *OEResidue* records. Secondary structure perception is performed by default when reading PDB files, but can be controlled with a new PDB input flavor, *OEIFlavor.PDB.SecStruct*.

5.73.3 OEGrid 1.4.0

New features

- *OEReadMTZ* has been added. This allows reading of standard MTZ crystallographic data files where the default MTZ column names from popular refinement packages were used.

5.73.4 OESystem 1.8.0

Minor bug fixes

- *OEColorGradientBase.GetColorAt* fixed to correctly round color components after interpolation.
- *OESystem::OEGeom3DGetOverlayRotTrans* renamed from `OEGeomGetOverlayRotTrans` and argument order changed to mimic other 3D geometry functions.
- The following *OEColor* functions and methods will automatically clamp color values to the range [0,255] and [0.0, 1.0] for integers and floating point numbers respectively:
  - *OEColor constructors*
  - *OEColor.SetR*
  - *OEColor.SetG*
  - *OEColor.SetB*
  - *OEColor.SetA*
  - *OEColorCreate*
  - *OEColorSet*

  A warning will be thrown to `OEThrow` if a clamping occurs.
- Valgrind warning silenced caused by mismatching `new []` and `delete` in OEChem's *small object memory allocator*. The proper `delete []` is now used though no real issue was observed by this bug in production.
- *OEInterface* no longer returns duplicate and confusing error messages by properly respecting the `error_level` argument in all failure conditions instead of using `OEThrow.Error`.
- `operator-=` and `operator+=` now function properly for negative numbers for *OEPredIter* and *OEConcatIter*.

5.73.5 OEPlatform 1.8.0

New features

- `OEPlatform::OEMallocaPtr::IsValid` added to provide a more straightforward way of testing whether the held pointer is `NULL`.

Major bug fixes

- *OEThreadLocal* will now properly destroy thread local variables upon thread termination for Visual Studio 2005 and later.
- *oeostream.operator<<* fixed to not crash when writing out a `double` that needs more than 80 characters to fully represent it.
- `OEPlatform::oeistream::pushline` method removed as it was a hack to implement `MOL2` file reading since the molecule delimiter comes at the beginning of the next molecule. *oeistream.getline* was the only other *oeistream* method that was aware of a `pushline` call. See the release note about the improvements to *OEReadMol2File* that justify the removal of this method.

Minor bug fixes

- *OEGetIPAddress* should be more robust for grabbing the IP address, not returning the loopback interface or inactive interfaces.
- *OEDirectoryScan* fixed to actually work on Windows.
- *OEThread* no longer has a copy constructor or `operator=` as these were dangerous and undefined.

Documentation changes

- The following chapters have been updated to new 2D depictions using the *OEDepict 2.x* library:
  - *Ring Perception* chapter
  - *Aromaticity Perception* chapter
  - *Connectivity Perception* chapter
  - *Stereochemistry Perception* chapter
  - *Pattern Matching* chapter
  - *Design Decisions* chapter
  - *Periodic Table Functions* chapter
- `OE3DToInternalStereo` documentation clarified to specify how it behaves in the absence of 3D coordinates.
- *Substructure Search* section fixed to more accurately reflect that subgraph uniqueness does not actually consider differences in matched bonds.

5.74 Version 1.7.7

5.74.1 OEChem 1.7.7

Minor bug fixes

- The *OELibraryGen* class checks the connectivity of each generated product. If any of the atom pairs of a product is connected by more then one bond, for example in molecule `C1C`, then that product is rejected and not returned by the `OELibraryGen.GetProducts` method.
- Fixed a bug that would invalidate an atom iterator after deleting the atom the iterator currently points at. This would only occur on molecules that already had deleted atoms when the iterator was constructed from a `GetAtoms` method that takes a predicate.
- `oemolstreambase.GetFlavor` will no longer crash when given a format larger than `OEFormat.MAXFORMAT`.
- `OEAddMols` will now set the dimension of the left hand side to the dimension of the right hand side if the dimension of the left hand side is zero. This allows an empty molecule to be used to accumulate 3 dimensional molecule without losing the dimension.
- Fixed a bug in the procedure for generating atom names for PDB files that could cause a crash when some atoms have very high atom indices.

5.74.2 OEBio 1.7.7

New features

- Added the `OESymmetryFractionalToCartesian` and `OESymmetryCartesianToFractional` functions to swap between crystal coordinates and cartesian coordinates.
- Added the `OEBuildUnitCell` function to apply the symmetry operations to the asymmetric unit cell to generate the complete unit cell given a molecule and a spacegroup.
- `OESetCrystalSymmetry` now has a force flag that sets the symmetry parameters even if the cell restrictions are violated.

Bug fixes

- *OECopyCrystalSymmetry* now follows standard OpenEye guidelines for argument position and naming to always have the output parameter first. This is an API breaking change, code that previously used *OECopyCrystalSymmetry* will no longer work.
- Slightly reduced memory footprint of symmetry operations.
- Crystal spacegroup names from PDB files are now handled more robustly when searching for CCP4 long and short names.
- Added symmetry operations for the P21/n space group.

5.74.3 OESystem 1.7.7

New features

- Adding the *OESystem.OETransparentColor* color constant that is used in **OEDepict TK** to generate PNG images with a transparent background.
- The `OEGeom*` functions should now be fully wrapped in the target languages: Python, Java, and C#.

Bug fixes

- The *OEPredIter* increment and decrement methods will now always move the underlying iterator before checking the validity of the underlying iterator. This fixed the OEChem bug that invalidated an atom iterator after the current atom was deleted.
- Fixed a memory leak inside *OEIter.Push* that takes a predicate.
- *OECopyBaseData* used to call `Clear` on the destination object before copying the data. This could clear an entire molecule instead of the just the generic data. *OECopyBaseData* will now only delete the existing generic data.

5.74.4 OEPlatform 1.7.7

New features

- *OEIsNormalNum* added to return whether a floating point number is normalized, denormalized, or zero.
- *OEGetProcessMemUsage* added to return the amount of memory being used by the current process.

Bug fixes

- *OEIsNaN* fixed to actually work when the compiler was passed `-ffast-math`. Also, it will now return `true` whenever passed `NaN` or `Infinity`.
- `OEAllocate` and `OEDeallocate` were deprecated long ago and are now removed.

5.75 Version 1.7.6

5.75.1 OESystem 1.7.6

Major bug fixes

- Internal debugging mechanism was left on in the previous release. This version turns it off improving performance across all libraries.

5.75.2 OEBio 1.7.6

Minor bug fixes

- As long as the *OEAssumption.BondedResidue* flag is turned off, *OEGetResidueAtoms* now correctly returns an iterator for every atom in a residue when some atoms in the residue are not connected to others.
- Because some crystal symmetry space-groups share the same space-group number which can lead to the use of the wrong space group operators, when possible the name of the symmetry is now used rather than the space group number.

5.76 Version 1.7.5

5.76.1 OEChem 1.7.5

Documentation changes

- *Generic Data handling* chapter added.
- All the `OEChem` examples have been overhauled. Several examples have been removed, new ones have been created and all the examples (across all the languages we support) have been reviewed to bring them up to a similar standard. For the details of example toolkit scripts see the *examples summary chapter*. Source code is also included in the documentation.

New features

- *OEAtomBondSet* container added.
- *OEIsAtomHybridization* functor added.
- *OEBase.GetDataIter* that takes an unsigned integer added for convenience.
- *OEIsMember* and *OEIsMemberPtr* can now be initialized with an *OEIterBase* pointer.
- *OEParseSmiles* function will now ignore extra leading spaces in non-strict mode.
- *OELibraryGen.SetTitleSeparator* will now store an internal copy of the string, fixing issues in wrapped languages.
- *OEIs3DFormat* function added as a convenience function for the user to determine whether a file format supports 3D coordinates.
- *OEMolBase.Clear* can now be called on a compressed DBMol and DBMCMol to wipe out the molecule entirely.
- *OEIsNonRingAtomDoubleBondedToRing* functor added to detect atoms double bonded to a ring. Most users consider atoms double bonded to a ring system as part of the "scaffold".
- MiniMols can now store coordinates, energy, and dimension. These are stored as generic data, so not particularly "mini". Just meant to allow MiniMols to be used as a more general purpose molecule implementation.
- Safety precautions to prevent corrupted molecules from being written were placed into the OEB writer for rotor-offset-compressed molecules. These molecules should not be modified, and definitely should not be written if they were modified.
- Both old format and new format PDB hydrogen atom names are now supported. The new PDB naming format is used when new H atom names are assigned. The call *OEResidueHydrogens* now supports an option to control which style of H atom names is assigned.
- Four new residue types have been defined: *OEResidueIndex.NHE* (carboxy-terminal amine) *OEResidueIndex.NME* (carboxy-terminal methyl amine) *OEResidueIndex.CIP* (AmberFF's +1 ion) *OEResidueIndex.CIM* (AmberFF's -1 ion).

Major bug fixes

- Fixed a crash in the OEB writer that occurred when very large values were set for the map index or isotope atom properties.
- Fixed a bug with parity bits in SDF files. This would occur if *OEPerceiveChiral* was called with an aromaticity model other than MDL before the molecule was written. The SDF writer now makes sure to re-perceive chirality based upon the MDL aromaticity model to get the correct parity bits.
- Fixed a memory leak that would occur when reusing a *oemolistream* for many different files under certain circumstances.
- *OEMol.operator=* fixed to not duplicate SD data when being assigned from an *OEGraphMol*.
- The FASTA reader now assigns the proper stereochemistry to the molecules it produces.
- The function *OEPerceiveResidues* now respects the following data preservation flags: *OEPreserveResInfo.Occupancy*, *OEPreserveResInfo.BFactor*, and *OEPreserveResInfo.FragmentNumber* both individually and implicitly as part of the flag: *OEPreserveResInfo.All*. Using the flag *OEPreserveResInfo.SerialNumber* will leave only the serial number of new atoms altered.

Minor bug fixes

- *OELibraryGen* fixed when explicitly defined bonds on the reactant side are not set correctly. This used to generate products with incorrect valence.
- *OEMiniMol::Sweep()* fixed to properly re-index atoms and bonds after some atoms and bonds were deleted.
- *OEKekulize* will now fail if the integer bond types are set correctly, i.e., not between 1 and 5 inclusive.
- *OEOFlavor.Generic.Default* will now be consistent in molstreams for all flavors. The *OEIsWriteable* function that takes a format and a flavor will now check the generic and specific flavor bits separately.
- *OERMSD* fixed when dealing with multi-conformer molecules with suppressed hydrogen atoms.
- *OERMSD* will now check whether the associated *OEMatchBase* actually belongs to the molecule being passed in.
- *OEParseSmiles* fixed to not crash when parsing empty strings in strict mode.
- *OELibraryGen.GetTitleSeparator* method signature changed to return a string directly. The previous signature and function couldn't operate and didn't actually do anything.
- *OESmartsLexReplace* fixed to not give incorrect SMARTS whenever a name appears twice in the bindings vector. Instead, the first pair is used, and a warning is thrown to indicate to the user about there being multiple pairs.
- *OEIsHeavy* is fixed to not return `true` for dummy atoms, that is, any atom with an atomic number of `0`. *OEIsHeavy* will only return `true` if the atomic number is greater than 1.
- *oemolithread.openstring* crash fixed when used without a valid format being set.
- Atom names perceived for unknown ligand types (*OEResidueIndex.UNK* and *OEResidueIndex.UNL*) are now constructed from the element name and an integer.
- *OEAddExplicitHydrogens* is fixed to produce better hydrogen placement geometries when adding hydrogens to sp3 carbons where the carbon's existing attachments vary significantly from ideal tetrahedral geometry.

5.76.2 OEBio 1.7.5

New features

- New *OEAltLocationFactory.GetCurrentLocations* methods provide a way to determine which alternate locations are selected. A new *OEAltLocationFactory.GetGroup* method is provided to convert from an *OEAltLocation* to an *OEAltGroup*.

- A new *OEGetRotamers* call that takes an *OEResidue* is now provided as a convenience.
- A new *OEExpandCrystalSymmetry* function expands the crystal symmetry of the source molecule into different conformers in the destination molecule.

Minor bug fixes

- `OEBio::OEHierResidue::GetOEResidue()` now returns a reference to a blank residue when the class is uninitialized instead of a reference to random memory.
- Adding *OESequenceAlignment.operator bool* method. An *OESequenceAlignment* object is considered invalid until its implementation is set. Both the *OEWriteAlignment* and the *OERMSD* functions checks whether the passed alignment is valid and throw a warning if it is not.

5.76.3 OEGrid 1.3.5

Minor bug fixes

- *OEGetCenterAndExtents* fixed to work on a *OESkewGrid* when it is regular grid, i.e. has no orientation.
- *OEErrorHandler.SetOutputStream* performance significantly improved to avoid quadratic behavior in certain circumstances.
- *OEWriteGrid* fixed to actually gzip the output if the file name ends in `.gz`. Before it would just write the file in the specified format, but never gzip it.

5.76.4 OESystem 1.7.5

New features

- *OEErrorLevel.Default* has been changed from *OEErrorLevel.Verbose* to *OEErrorLevel.Info*. This allows `OEThrow.Verbose` to be used for printing more information to the terminal, but it will only appear when `OEThrow.SetLevel` is called with the *OEErrorLevel.Verbose* level. In the past, the *OEErrorLevel.Verbose* level was essentially ignored since it was nearly the same as *OEErrorLevel.Info*.
- *OELinearColorGradient* class added to interpolate colors between specified ranges. *OEColorStop* class added to initialize *OELinearColorGradient* objects.
- `OESystem::OEBkgrdGrey` added as a new color for using a light grey background of an image.
- *OECopyBaseData* convenience function added for copying *generic data*.
- *OEGetTag* heavily optimized for multi-threaded use.

Minor bug fixes

- *OEBitVector.SetData* crash fixed where it would try to copy more bytes than the actual size of the given data.
- *OEStringToNumber* function now allows for new-line characters, `\n`, to appear after the digits in the string.
- *OEBase generic data* can now handle an empty `std::vector` through OEB reading and writing.
- *OEStringTokenize* now has an additional parameter, *concat_delim*, that specifies whether consecutive delimiters should be treated as a single delimiter. This is useful for parsing CSV files with empty fields. However, the default behavior is to behave as before, treating consecutive delimiters as a single delimiter.
- *OEGridBase.GetXIdx*, *OEGridBase.GetYIdx*, *OEGridBase.GetZIdx* fixed so that the dimensions are guaranteed to always be within the dimensions of the grid. These methods would sometimes return indices that were outside the range of the dimensions of the grid due to floating point rounding errors.
- `unsigned` can now be used as the `!TYPE` in interface files for unsigned int parameters.

- *OELinearInterpolate* performance has been significantly improved.

5.76.5 OEPlatform 1.7.5

Minor bug fixes

- `OEEOF` is now defined by the `OEPlatform` headers and is the value returned by `oestream` methods to indicate that the end of the file has been reached.
- *oeostream.operator<<* can not properly write a `size_t` on 64-bit Windows.

5.77 Version 1.7.4

5.77.1 OEChem 1.7.4

New features

- The substructure search algorithm was optimized resulting in a significant performance increase for certain queries.
- Adding CIP (Cahn-Ingold-Prelog) atom and bond stereo perception.

> **See also**
>
> – *OEPerceiveCIPStereo*
>
> – *OESetCIPStereo*
>
> – *CIP Stereo Perception* section

- Adding two new options to the *OEMDLQueryOpts* namespace.
  - The *OEMDLQueryOpts.MatchAtomStereo* option allows R/S atom stereo configuration match.
  - The *OEMDLQueryOpts.MatchIsotope* option allows isotope match.
- The CDX file reader was extended to read superatoms. Superatoms such as (Me, OEt, Ph) are represented in CDX files as disconnected fragments with no coordinates. OEChem now reads superatoms correctly, although no coordinates are assigned for the fragments.

Major bug fixes

- The default bfactor of OEResidues was changed to 20.0 from 0.0. This is to allow OEResidues written out of *OEChem* to play nicer with other software that doesn't know what to do with a bfactor of 0.0.
- *OEDBMCMols* would crash during copy construction if already compressed. Allowing `OEDBMCMols` to copy construct while compressed allows them to be directly used inside of STL containers.

Minor bug fixes

- Bug fixed in the MDL query interpretation (*OEBuildMDLQueryExpressions*) that occurred when a hydrogen atom occurred in a generic atom list of a ring query atom.
- Bug fixed that occurred when "zero-level" parentheses are used in SMARTS to perform component queries. The `(X).(Y)` SMARTS means that the `X` and `Y` SMARTS pattern have to be located in separate components of the target structure. For example `(CO).(C=O)` has a match in `CCO.CC=O` but not in `OCCC(=O)`.
- Fixed bug that broke reading of any map idx greater than 9 from OEB files.

- Fixed a rare crash when using DBMols from Python or Java.
- Fixed a bug where MDL parity bits were ignored when reading a mol file with no dimension set.
- Modified the coordinate handling in CDX to invert and scale coordinates correctly.

5.77.2 OESystem 1.7.4

New features

- Many usability improvements and a new documentation chapter, *Command Line Interface Definition*, on using *OEInterface* for command line parsing.
- Adding new *OEColor* constants such as *OESystem.OEBrown* and *OESystem.OEViolet*. For the full list see the *OESystem* namespace.
- Adding the *OEGeom3DMatrixTranspose* and *OECenter* OEMath functions.
- Major optimization to *OEAnnotation.AddChild*.

5.77.3 OEPlatform 1.7.4

Minor bug fixes

- Fixed a bug where some operations on an oeigzstream (including molstreams on gzipped files) could result in an infinite loop when the associated gz file was prematurely truncated.

5.77.4 OEBio 1.7.4

New features

- Added a new overload of *OEExpandCrystalSymmetry*.

Minor bug fixes

- *OEGetSpaceGroup* now properly trims the input string enabling better matches.
- *OEGetCrystalSymmetry* now properly assigns the spacegroup number.

5.77.5 OEGrid 1.3.4

New features

- Added the `OESystem::OEQuadraticInterpolationAndDerivatives` function.

Minor bug fixes

- Fixed bug where grids were not written out when attached to molecules. This could only sometimes occur in a statically linked C++ application depending on what set of **OEGrid** functions were being used.

5.78 Version 1.7.3

5.78.1 OEChem 1.7.3

This was a limited in-house release.

5.79 Version 1.7.2

5.79.1 OEChem 1.7.2

- `OEPDBIFlag`, `OEPDBOFlag`, `OEMDLOFlag`, `OEMOPACFlag` constants namespaces are deprecated.

New features

- *OEReadMDLQueryFile* now sets the title of the query molecule.
- *OEChem* now recognizes the MDL "wavy" bond. An *OEBondBase* can be queried for its stereo type by querying its generic data for the *OEProperty.BondStereo* tag. The returned `unsigned int` will be from the *OEBondStereo* namespace.
- *OEChem* now supports MDL query file R group definitions. These are lines starting with `M RGP`. The R group is represented as a *OEQAtomBase* with a *map index* the same as the R group number in the file.
- *OEChem* will now correct amidine and guanidinium functional groups where the bonds between C and N are marked as aromatic in MOL2 files.
- Significant performance improvement to reading single-conformer molecules from OEB files. Other minor performance improvements were made to OEB as well (including multi-conformer molecules). This speeds up `OEDBMol` compression and uncompression as well.
- Removed `OECopyHistory` function. `OEChem::OEReadHeader` is now smart enough to read the header into the *OEHeader* history if it is already populated with another header's information.
- *oemolthreads* now support reading and writing *OEHeader* objects.
- Added the *oemolthreadbase.PeekMol* method.

Major bug fixes

- Fixed a *OEReadMDLQueryFile* segmentation fault when perceiving aromaticity with spirane rings containing generic atoms.
- Fixed a bug where atom parity was ignored when reading a 3D MOL file.
- Fixed a memory leak when initializing an *OEMCSSearch* with its *constructor*. The workaround in 1.7.0 is to use the *OEMCSSearch.Init* method instead.
- Fixed a memory leak with *OEMCSSearch* when *OEMCSSearch.SetMaxMatches* was set to a number lower than the total number of matches found.
- Fixed memory leak when creating and destroying multiple *OELingoSim* objects.
- Some *OEOFlavor.PDB* flavor combinations would generate corrupt data.
- *OESmilesAtomCount* now includes atoms of the form `[#6]` in its count.
- Fixed a bug where map indices on explicit hydrogens would be ignored when generating SMILES. For example, `[CH]` would get generated even if the hydrogen had a map index specified. Now `[H:1][C]` will be generated.

Minor bug fixes

- Energy set on an *OEGraphMol* using *OEMolBase.SetEnergy* would not get written out to OEB files. Energy data will now round-trip through the OEB format.
- Automatic conformer combining through the *oemolistream.SetConfTest* interface will now combine separate single-conformer molecules in an OEB into a multi-conformer molecule similar to the other file formats *OEChem* supports for that behavior (MOL2, SDF, etc).
- *OEPerceiveBondOrders* is more stable to pre-existing bond orders on the molecule.

- Rare non-deterministic bug fixed for sulfur monoxide bug in *OEPerceiveBondOrders*.
- *OEChem* now recognizes the DU residue name and the mythical DT residue which is synonym for `` T``.
- The new overload of *OEIsReadable* to take a string would return `true` if there was no periods ('.') in the string and the string began with a readable file extension, for example, "MDL-FOO".
- Fixed a bug when *OELibraryGen* failed to generated products with correct Kekulé-form.

5.79.2 OESystem 1.7.2

New features

- Added the *OEGetBitCounts* function to efficiently get certain bit counts when comparing two *OEBitVector* objects. This function is useful for implementing custom fingerprint similarity measures.
- Added the *OEBitVector.SetData* method to allow a rapid method of initializing an *OEBitVector* with binary data.
- Added *OEErrorHandler.Verbose* method to send *OEErrorLevel.Verbose* level messages.
- Added `OESystem::OEHeader::operator bool` to test whether the *OEHeader* has any data in it.
- Added *OEBoundedBuffer.Peek* and *OEProtectedBuffer.Peek* methods.
- Optimized writing generic data attached to an *OEBase*. This optimized `OEDBMol::Compress`.
- Minor optimization to a common *OEChem* iterator.

Major bug fixes

- Fixed a memory leak with `OEConstIter`.

Minor bug fixes

- Renamed the `OEMakeFP` function to *OEParseHex* to be more explicit about what is does.
- Valgrind would throw warnings about a change made in 1.7.0 for thread safety issues. Though this was not a true memory leak the code has been slightly changed to silence this warning.
- Fixed an exponential growth bug when adding history to an *OEHeader*.
- Assignment operators for *OEMultiGrid* now properly copy the titles of the subgrids.

5.79.3 OEPlatform 1.7.2

New features

- Added the *OESetLicenseFile* function to allow dynamically setting which OpenEye license file to use.
- Added the *OEGetCurrentWorkingDirectory* function to return the current working directory for the process.
- Optimized *oeigzstream.seek* to only rewind the stream if seeking backwards. Therefore, the optimal way to seek in a gzipped file is in increasing `oefpos_t` order.
- Optimized *oeisstream.getbuffer*.
- Optimized *oeosstream.clear*.

Minor bug fixes

- Revised the *oestream.length* and *oestream.size* documentation to be more accurate.
- `oefpos_t` now works for files greater than 4GB on 64-bit windows.
- Fixed an infinite loop that would occur from the following code:

```
oeosstream sfs;
sfs.open("foobar");
sfs.write("blah");
```

  oeosstream.open will return `false`. Ignoring the `false` return value would cause *oeosstream.write* to loop indefinitely.

5.79.4 OEBio 1.7.2

New features

- Introduced new *OEBio* classes to manage alternate locations in protein data bank structures along with a new PDB input flavor *OEIFlavor.PDB.ALTLOC* to retain alternate location atoms on input.

> **See also**
>
> *Alternate Locations*

Major bug fixes

- The crystal symmetry routines were broken in the 1.7.0 release by a corrupted matrix.
- When reading symmetry from external sources (*i.e.* PDB files or maps) a warning is thrown when reading out of date space groups rather than fail to read symmetry.
- Setting space groups will fail unless the most current space group constraints are used.
- Added the following space group aliases for older style space groups.

```
I 1 2 1 -> C 1 2 1
P 1-    -> P -1
```

5.79.5 OEGrid 1.3.3

New features

- *OEMakeGridFromCenterAndExtents* added for a grid of an arbitrary template type.

Major bug fixes

- Fixed a rare crash in *OEMakeRegularGrid* that was caused by floating point round-off error.

5.80 Version 1.7.1

5.80.1 OEChem 1.7.1

This was a limited Java only alpha release.

5.81 Version 1.7.0

5.81.1 OEChem 1.7.0

New features

- Canonical isomeric SMILES generation has been significantly improved (*OECreateIsoSmiString*). On a test set of 9,962,003 compounds (4,025,817 with atom or bond stereo) *OEChem* 1.6.1 would generate different canonical isomeric smiles for 135,985 of the compounds based on random reordering of the atoms. This failure rate has been reduced to just 78 compounds, a 99.94% improvement. Furthermore, the generation has been optimized so that it is roughly 10-30% faster than the *OEChem* 1.6.1 algorithm.
- *OEReadMDLQueryFile* has been added to read MDL query files into the *OEQMol* object. This allows for easy integration of MDL query files with the swath of *OEChem* tools based upon query molecules. The MDL query based substructure search was tested on a set of 655 query files.

> **See also**
>
> The *Substructure Search with MDL Queries* chapter in the *OEChem* theory manual.

- *OEChem* also supports MDL reaction based library generation. Reaction file can be imported into a *OEQMol* object by calling *OEReadMDLReactionQueryFile* function. The *OELibraryGen* object then can be initialized with the imported reaction. The library generation was extensive tested on a set of 160 diverse reactions.

> **See also**
>
> The *MDL Reaction Query File* section in the *OEChem* theory manual.

- The *MiniMol* implementation of *OEGraphMol* has been made more robust and optimized significantly for both speed and size. There used to be an arbitrary 1000 atom and bond limit, the limit is now a lot higher, 2^{15}. This implementation requires *Compress* be called on the molecule after construction to maximize space efficiency, but does not require *UnCompress* be called on it before it can be used. This makes it an ideal molecule implementation for in-memory substructure searching.
- Added *oemolithread* and *oemolothread* for threaded molecule I/O. *OEReadMolecule* and *OEWriteMolecule* are thread-safe on *oemolithread* and *oemolothread* respectively.

> **See also**
>
> The *Input and Output Threads* section in the *OEChem* theory manual.

- Added implementation of Zap 9 Radii from [Nicholls-2008] through the *OEAssignZap9Radii* function.
- Added *OEShortestPath* function since it is an often-asked-for algorithm.
- Added *OEIdxSelected* predicate for easy sub-setting of a molecule using an array of `bool` indexed by *indices*.
- Added *OECount* function for easily counting atoms or bonds in a molecule based on arbitrary *predicates*.

Major bug fixes

- Fixed seg fault in SMARTS parsing when the SMARTS contained the [<atomic mass>H<charge>] combination.
- Fixed `invalid address alignment` crash in the sketch file reader on Sparc.
- SD tag names were limited to 75 characters. This release raises the limit to 4096.
- Several major changes are made in the library generation process in order to ensure that products are generated with a valid Kekulé form. If there is an explicit hydrogen in the product side of the input reaction, then *OELibraryGen* will add hydrogens to the generated products accordingly and the first Kekulization will be based on this reaction specification. If it is unsuccessful, *i.e.* *OEKekulize* returns `false`, then alternatives are tried by adding and removing implicit hydrogens from specific atoms until a valid (but arbitrary) Kekulé form is identified.

> **See also**
>
> The *Product Kekulization* section in the *OEChem* theory manual.

- Significant number of improvements were added to *OEChem*'s PDB file parser (*OEReadPDBFile*) in the area of atomic number determination. By default PDB atomic symbol field is used to determine the atom type. However, for a subset of "known" residues, including all of the amino and nucleic acids, we continue to use the atom name heuristics, which for this subset are more reliable. Currently, there are no conflicts/discrepancies between the PDB atomic symbol and the atomic number we perceive in the PDB file reader in the entire wwPDB repository.
- The refinement of the hydrogen placement method (*OESet3DHydrogenGeom*) improved numerical stability and increased speed for most simple placement operations, special heuristics for carboxylic acids, toluene-like methyl rotors attached to aromatic rings, fleeing heuristics for alkanes and perpendicular support for allenic systems. Additionally, hydroxyl rotors are placed using a quick local scan for strong acceptors (alpha acceptors) within a 3.0 Angstrom radius. More hydrogen bond lengths data specifically to As, Ge, Se and Te was also added ([Sutton-1958]).

Minor bug fixes

- *OEGraphMol.operator=* does the same thing as the *OEGraphMol* copy constructor. This allows *OEGraphMol* to be used directly inside STL containers without losing the selected *OEMolBase* implementation. Previous versions of *OEChem* would change the molecule implementation unexpectedly if the STL container needed to relocate the object in memory. However, the following code changes meaning with the switch from 1.6.1 to 1.7.0:

```
OEGraphMol gm1;
OEGraphMol gm2(OEMolBaseType::OEDBMol);
gm1 = gm2;
// gm1 is now an OEDBMol implementation,
// in 1.6.1 it would be the default implementation
```

- The *OEMolBaseType.OEDBMol* implementation can now copy construct in compressed mode.
- *OEReadMolecule* would sometimes return `true` when an empty molecule was present in the input file. The high level *OEReadMolecule* function will never return `true` when the molecule does not contain any atoms. Low-level molecule routines should be used if empty molecules are desired.
- Stabilized the atom output order from successive calls to *OEWriteMolecule* to MOL2 when the molecule contained residues.
- Deprecated the non-OE prefixed `SmartsLexReplace` function name. Renamed to *OESmartsLexReplace*.
- Fixed the bug in substructure search that occurred when *OESubSearch* was initialized with a SMARTS string starting with a hydrogen atom (such as `[#1]O[C,N,S,P]=O`). The order of the atoms were reordered even when allowReorder parameter was set to be `false`.
- When initializing an *OESubSearch* object with a SMARTS pattern the 'reorder' parameter is changed to `false` by default. This parameter is currently ignored *i.e* the atom order in the returned matches is always identical of the atom order in the SMARTS pattern.
- Fixed a bug in `OEGetAromatic` and `OEGetBondOrder` to retrieve aromaticity/bond order from query expressions.
- A restriction is added to the interpretation of SMIRKS in 'strict' mode. This requires that all atom maps in the reaction have to be pairwise when *OELibraryGen* is initialized with a SMIRKS string or *OEQMolBase* object.
- DBREF, SEQADV, MTRIX1, MTRIX2 and MTRIX3 pdb data lines are now kept when parsing the file. The data can be accessed by *OEGetPDBData*.
- Fixed the bug in *OEWritePDBFile* that caused writing the MODEL number into a wrong position in a pdb file.
- Fixed the bug in the SMILES canonicalization process for special cases when the input SMILES contains R-Group information (such as `[R1]c1ccc(cc1c2cccc(c2)[R3]`).
- Tweaked the *OpenEye* charge model such that the three valent Beryllium has a negative one charge. This makes `F[Be-](F)F` equivalent to the charge separated form `[Be+2].[F-].[F-].[F-]` fixing 61 ligands in the PDB data set.
- Improved the bond order perception (*OEPerceiveBondOrders*) support:
  - for arsenic acids, including the 'cacodylate' ion (example in 3DUE pdb).
  - for uric acid and related heterocycles.
  - for azobenzenes and similar compounds. All acyclic 'nitrogen(2)-nitrogen(2)' bonds now undergo a strict distance check independent of the (single) bond angle at each end. This corrects 1SRE, 1SRF and 2GBY pdb entries.
  - for benzoquinones and anthraquinones.
- Addressed the problem in *OEAssignAromaticFlags* that caused long processing times when reading (*OEReadMDLFile*) on some pathological pseudo-fullerenes.
- Added PDB support for the following:
  - sidechain recognition for the RNA residue 'YG' and 'H2U'
  - naming of PDB residue 'BME'
  - the N-terminal modification 'FOR'
  - the cofactor 'FMT' (which is "formic acid" or "formate")
- The following problems were fixed in the PDB file parser (*OEReadPDBFile*):
  - 'anomalous mercury' problem for the residue 'DVA' (example in 2IZQ pdb)

  - spurious Holmium problem (in residues ‘CEH’ and ‘NGR’)
  - naming of the ‘ P ‘, ‘ O1P’, ‘ O2P’ and ‘ O3P’ atoms in the nonstandard PDB residues ‘PTR’, ‘SEP’ and ‘TPO’.

- Improved the initial partial charge parameterization (*OEMMFF94InitialCharges*) for selenium (atom type 83).
- Improved the perception of reactions in ISIS Sketch files. Most importantly we now support the ‘rxnarrow’ object generated by recent versions of ISIS, such as MDL Draw and Symyx Draw. We also now allow the sketch to contain multiple lines, provided that only one has an arrow, and to allow the arrow direction and arrow coordinates to be specified in arbitrary order.

5.81.2 OEGrid 1.3.2

New features

- The user is no longer required to call *OEInitGridHandlers* in order to attach grids to molecules and then write them out to OEB. This occurs at library link time now.

> **See also**
>
> The *Generic Data* section in the *Grids* chapter for details of attaching grids to molecules.

Major bug fixes

- *OEReadGrid* would crash on gzipped files. It now properly uncompresses the data before reading it into a grid.

5.81.3 OESystem 1.7.0

New features

- *OERandom* can now produce random integers through the *OERandom.NextInt* method.
- Added an *OEBitVector* constructor which takes an *OERandom* for generating random bit strings. Added *OEBitVector.operator<* method so *OEBitVector* can be used in STL containers.
- Added *OEBoundedBuffer* and *OEProtectedBuffer* objects useful for communicating between threads in multi-threaded applications.
- Added *OEWallTimer* since *OEStopwatch* actually reports CPU time so if multiple threads are being used it reports the added CPU time from each thread. Also added *OECycleTimer* for doing high precision timing based on clock cycles. For example, on x86 this uses the `rdtsc` instruction and is thus susceptible to its trade offs in comparison to using CPU time from *OEStopwatch*.
- Added *convenience constructor* to *OEInterface* for the most common use case. Where a commandline needs to be parsed relative to a particular interface definition.

Major bug fixes

- Deprecated `OESetThreadSafe` as it only gave the illusion of thread safety to the *OEChem* toolkits. Use *OESetMemPoolMode* instead as outlined in the *Memory Management* section. Most users should just ignore this issue altogether, as the defaults are sufficient (and have been optimized in 1.7.0). The user should only consider calling *OESetMemPoolMode* when passing *OEChem* objects between threads.
- Fixed regression where *OEBitVector.FromHexString* would no longer recognize hex strings in lowercase. It should also be noted that *OEBitVector.ToHexString* encodes the fractional length of the last 4 bits as the last character.

Minor bug fixes

- Added *OEErrorHandler.Debug* as another output option to match *OEErrorLevel.Debug*
- *OEErrorHandler* now properly does nothing when *OEErrorLevel.Quiet* is passed.
- Added *OEUnaryTrue* and *OEBinaryTrue* to match their inverses that already existed.

5.81.4 OEPlatform 1.7.0

New features

- Added *OELock* class to do scoped locking around *OEMutex* objects. This ensures mutexes are released in the event of stack unwinding, such as when exceptions are thrown.
- Added the following cross-platform threading primitives: *OEThread*, *OEThreadLocal*, *OECondition*, and *OEOnce*.
- Added *OEGetTimeOfDay* to wrap gettimeofday on posix, but provide our own implementation on Windows since it doesn't exist on Windows.
- Added *OEGetNumProcessors* to return the number of cores available on the system.

Major bug fixes

- Fixed rare off by one buffer overflow in *oeogzstream* initialization.
- Fixed `seek` and `tell` on files greater than 4 gigabytes on Windows.

Minor bug fixes

- The *OEMutex* implementation would default to a no-op implementation even when a valid mutex implementation was available through pthreads and compiling with a non-GCC compiler.
- *oeifstream* will return `false` when *oestream.seek* is called and return `0` when *oestream.size* is called when the stream points to `stdin`.
- Corrected instances where streams were using `oefpos_t` for memory operations and `oesize_t` for file operations. The rule is memory operations should use `oesize_t` (*e.g.* *oeistream.read*), and file operations should use `oefpos_t` (*e.g.* *oestream.seek*).
- Changing `unsigned int` arguments to `oesize_t` for *oeisstream* to be able to handle memory larger than 4 gigabytes on 64-bit machines.
- The following functions were global scope: *OEGetIPAddress*, *OEGetHostIdent*, *OEGetDomainName*, and *OEGetHostName*. They are now located in the *OEPlatform* namespace.

5.82 Version 1.6.1

5.82.1 OEChem 1.6.1

New features

- `RegisterMolParameters` renamed to *OERegisterMolParameters*. However, the function is now automatically called at link time alleviating the user from having to call this function in order to use molecules with *OEInterface* s.
- *OEReadXYZFile* uses the "Unichem" numeric convention for elements that don't have a one or two-character IUPAC symbol.
- Adding `const string&` variants of *OEParseSmiles* and *OESmilesAtomCount* functions to the *OEChem* API.

Major bug fixes

- *OEPerceiveBondOrders* function is improved by:
  - distinguishing unsaturated allene (C=C=C) from propyne (C#CC)
  - assigning the double bond to a shorter of the bonds to a terminal oxygen in functional groups such as *C(=O)O or *C(=O)[O-]

Minor bug fixes

- Hydrogen bond lengths for As, Ge, Se and Te elements are added to improve the 3D coordinates assigned for explicit hydrogens by *OESet3DHydrogenGeom* function.
- Optimization of OEChem::OEDBMol compression.
- Memory leak was fixed in *OERMSD*.
- Bug was fixed in the binary write routine of *OEGraphMol* and *OEMolBaseType.OEDBMol* parameters, that caused problems during a multiprocessor run.
- Memory problem that occurred when reading molecules with large number of conformations was fixed by allocating memory from the heap rather than stack in such cases.
- Using *operator=(const OEMolBase &)* on itself will no longer crash.

5.82.2 OESystem 1.6.1

New features

- New constructor added to the *OEInterface* class, that takes the interface data along with argc and argv arguments of main(). Using this constructor, an *OEInterface* object is configured, parses the command line, and prints the help message if requested.

Major bug fixes

- The direction of rotation matrix given by *OEGeom3DEulerToRotMatrix* and *OEGeom3DQuaternionToRotMatrix* functions is now consistent with the other geometry handling routines.
- Memory leak, that could occur in multi-threaded environment (such as Java), was fixed.

Minor bug fixes

- *OEParseCommandLine* prints more sensible error message.

5.82.3 OEPlatform 1.6.1

Minor bug fixes

- Retrieving the host identifier for MAC Darwin platform is improved.

5.83 Version 1.6.0

5.83.1 OEChem 1.6.0

New features

- *OEChem*'s PDB residue perception code now follows the changes required by the PDB version 3.0 standard. This includes disambiguation of the DNA residues "DA", "DC" and "DG" from the RNA residues "D", "C" and

"G". Nucleic acid backbone atom names now end in an apostrophe instead of a star/asterisk. The default ligand name in *OEChem* is now "UNL" instead of "MOL" or "LIG".

- There have been significant improvements to *OEChem*'s bond order perception code including phosphates, thiophosphates, dithioic acids, oximes, aldoximes, sulfur oxides, sulfites and iron-sulfur clusters.
- The cis/trans detection logic in `OE3DToBondStereo` has been robustified to do better with 2D depictions containing bonds that are co-linear with a chiral double bond or have zero length. The code now continues searching for additional incident bonds that have non-zero length and aren't co-linear.
- Significant improvements have been made to *OEChem*'s DNA/RNA perception code. The code now handles/recognizes truncated RNA biopolymers, and recognizes the bases "1MA", "2MC", "5MC", "5MU", "7MG", "M2G", "OMG", "OMC", "PSU", etc…
- *OEChem*'s residue perception code now handles/recognizes a much larger set of common co-factors and ligands, including "ADP", "ATP", "DMS", "EDO", "FAD", "HEM", "NAD", "NAG", "PEO" etc…
- *OEChem*'s residue perception code now handles the nonstandard amino acid ornithine (ORN).
- A new `OEAssignFormalCharges` function variant has been added to the *OEChem* API to allow assignment of formal charges on a specific *OEAtomBase* pointer. The existing `OEAssignFormalCharges` function continues to operate on the whole molecule.
- A new `OEPreserveResInfo.AtomName` flavor has been added to the `OEPreserveResInfo` namespace to allow the `OEPerceiveResidues` function to preserve the original PDB atom name.
- The *OpenEye* formal charge model has been extended such that a four-valent aluminum (aluminium) now has an implicit negative charge, and aluminum ions have a +3 formal charge. The model has also been tweaked to consider the sulfurs in iron-sulfur clusters as neutral `[S]` radicals.
- The *OpenEye* hydrogen count model has been tweaked to prefer five-valent phosphorus, such as `O=[PH2]O` over three-valent `O=PO`.
- The values returned by `OEGetAverageWeight` have been updated and revised to follow the latest (2007) recommendations of the IUPAC Commission on Isotopic Abundances and Atomic Weights.
- The `OEParseSmarts` and `OEParseSmirks` functions have been enhanced to allow a TAB character `\t` to be treated as a separator after a SMARTS pattern. This matches the behavior of the SMILES parser, `OEParseSmiles`, and simplifies the task of writing "patty"-like applications.
- The *OEChem* SMILES and SMARTS parsers have been tweaked to allow the backslash used in specifying cis/trans stereochemistry to be duplicated in the input string, *i.e.* `C\\C=C\\C` is now interpreted as `C\C=C\C`. This is convenient when working with programming languages such as C and C++ where the backslash is used as an escape character. Embedding SMILES in C/C++ source files requires the strings look like `C\\C=C\\C` which previously couldn't be cut'n'paste like regular SMILES strings.
- The interpretation of acyclic aromatic elements by *OEChem*'s SMILES reader now more accurately follows the Daylight toolkit. For example, `n` is interpreted as `[NH2]` and not `[N]`, and `nn` now means `N=N` instead of `[N]=[N]`.
- Fixed an obscure corner case in the *OEChem* SMILES writer, when not performing aromaticity perception and using the low-level SMILES writer. We need to preserve the explicit single bond (hyphen/minus) in `[cH2]-[cH2]` otherwise `[cH2][cH2]` would get interpreted like `cc` and result in `c=c` and `[cH2]=[cH2]`.
- The MDL file format reader has been enhanced to allow TABs in addition to spaces as separators in `M CHG`, `M RAD` and `M ISO` lines.
- The Sybyl `.mol2` file format reader has been enhanced to recognize pyrylium-like ring systems, containing charged oxygen atoms. A minor bug has also been fixed that could assign inappropriate formal charges to substituted nitrates.

- In Sybyl `.mol2` format files, the atoms types `d` and `t` are now treated identically to `D` and `T` and interpreted as Deuterium and Tritium respectively. Previously, they'd be interpreted as hydrogen atoms, but the isotope specification wasn't getting set.
- The Tripos bond types in Sybyl .mol2 format files are now treated as case-insensitive. We now treat `AR` and `Ar` as identical to `ar`, and `AM` and `Am` as identical to `am`, etc…
- The CambridgeSoft CDX file format reader has been significantly rewritten to address bugs in the reading/writing of 3D coordinates.
- The *OpenEye* OEB file format reader is now more robust to invalid, corrupt and/or truncated input files.
- Added versions of *OEIsReadable* and *OEIsWriteable* that can directly take a filename or extension.
- Significant work was done on SD data robustness.
  - If SD data was attached to an *OEMCMolBase* and an *OEConfBase* then written to OEB and read back into an *OEMolBase* the data from the *OEConfBase* would appear to disappear. This would result in losing the data if then written to SDF.
  - SD data attached to an *OEMolBase*, written to OEB, then read into an *OEMCMolBase* will now be attached to the *OEConfBase* instead of the *OEMCMolBase*.
  - Significant speed improvements were made to SD data (*e.g.* csv2sdf saw a 4-fold speed improvement).
  - See *Dude, where's my SD data?* in the programming theory manual for more details.
- The *OEReadPDBFile* and *OEWritePDBFile* functions are able to read and write ANISOU records, respectively. ANISOU records, which are atom property representing anisotropic temperature factors in PDB, are scaled by a factor of 10^4 and represented as integers.
- *OEGetCenterOfMass* function, which computes the center of mass of a molecule (with or without atomic weights), was added to the `OEChem` namespace.

Major bug fixes

- The algorithm that generates canonical SMILES did not ignore cis/trans stereo hydrogens and produced `[H]N=CC`, rather than the correct `N=CC` canonical SMILES.

> **Warning**
>
> Even though this bug fix has affected only a small percentage of canonical SMILES, we highly recommend the regeneration of all canonical SMILES.

- Small improvements have been made to the generation canonical isomeric smiles.
- A problem has been fixed in *OEChem*'s Kekulization algorithms for large molecules (with between 250 and 1000 atoms) that can't be assigned a valid Kekulé form. The changes in *OEChem* 1.5.0 that attempted to assign as much of a Kekulé form as possible upon failure could occasionally lead to *OEKekulize* returning `true` for an invalid molecule.
- A performance problem in *OEChem*'s aromaticity perception has been resolved. Previously pathological substituted fullerenes and PAHs could cause *OEChem*'s aromaticity routines to take over a minute to perceive all of the conjugated cycles. Algorithmic improvements to *OEChem*'s aromaticity perception now allow all of the reported cases to be processed in a fraction of a second.
- A rare problem interpreting the stereo from wedge/hash bonds around atoms of degree three has been resolved. When we have two bonds in the plane, and the third marked as a wedge or a hash, we need to determine whether the raised/lowered bond is in the larger or smaller sector subtended by the two in-plane bonds. A bug in this code failed to handle the case when all three bonds lay in the same half-circle. This problem is extremely rare, for

example, no cases were found in the 250,251 MDL connection tables distributed by the NCI as the NCI August 2000 database.

- Memory leak problem was fixed in *OELibraryGen*.

Minor bug fixes

- The *OEChem* MDL mol file reader has been improved to allow the dimension field in the connection table header line to be omitted, and still correctly decide whether to process wedge/hash bonds or determine chirality from 3D coordinates. Previously, the molecule's stereochemistry would be set incorrectly if the optional header line was missing.
- The MDL file reader now perceives aromatic cycles using the MDL aromaticity model prior to calling `OEPerceiveChiral`. This ensures that alternate Kekulé forms of substituted phenyl rings (for example) don't inappropriately split symmetry groups, causing achiral double bonds to acquire specified cis/trans stereochemistry.
- An aesthetic improvement has been made to the rules used in the `OEMDLPerceiveBondStereo` function that assigns wedge and hash bonds to depictions. For acyclic bonds, we now prefer to place the wedge or hash on bonds to non-ring atoms. A typo in the previous rules reversed this priority.
- The *OEChem* SMILES writer was being miscompiled on IBM AIX 5.x resulting in canonical SMILES that differed from those on other platforms. The code has been rewritten to avoid the issue in IBM's xlC compiler, so the SMILES are once again identical to the other platforms.
- The *OEChem* PDB file parser has been updated to reflect the latest atom name exceptions in the RCSB/wwPDB database. These changes should eliminate the spurious Holmium and Helium atoms perceived in recently added ligand residues.
- Numerous small performance improvements have been made to *OEChem*.
- The torsion cutoff values for perceiving cis/trans bond stereo from 3D are relaxed in `OE3DToBondStereo` function. The cis cutoff is increased to 30 from 15, the trans cutoff is lowered to 150 from 165.
- Even when the maximum number of matches is set, the MCS search can not be terminated upon reaching this limit, since there is no guarantee that the maximum common substructure has been detected. Instead, the search continues, then the best *N* matches are returned, where *N* is set by `OEMCSSearch.SetMaxMatches`.
- The exhaustive and the approximate MCS algorithms no longer use different functions to determine whether a match is unique or not. Several other small problems were fixed in order to ensure that all matches located by the approximate method are also detected by the exhaustive one.
- After changing the atomic number of an atom with `OEAtomBase.SetAtomicNum`, the aromaticity and chirality of the molecule have to be re-perceived with the `OEAssignAromaticFlags` and `OEPerceiveChiral` functions, respectively.
- Improved the stability of `OEPerceiveResidues` to not reorder atoms.
- An API point was added to *OELibraryGen* to change the character used to separate product molecules title when concatenating reaction molecule titles together. For more information see `OELibraryGen.SetTitleSeparator` and `OELibraryGen.GetTitleSeparator` in the API manual.
- A rare problem occurred in the substructure search when hydrogen atoms were matched first. This problem has been solved by rearranging the order in which atoms are taken into consideration, moving hydrogens to the end of the match order. Other small modifications have been made to improve the performance of substructure search.
- `OEFindRingAtomsAndBonds` is automatically called to perceive rings in structures returned by *OEUniMolecularRxn* or *OELibraryGen*.
- Bug was fixed that caused *OEGraphMol* and *OEMol* parameters to always fail to load in `OERegisterMolParameters`.

Changes in documentation

- *OEDeleteSDData* was improperly documented in the theory manual. The old documentation stated that only the first instance of a tag was deleted when all instances of the tag were actually deleted. The documentation has been corrected to state that all instances of a tag are deleted.
- *Maximum Common Substructure Search* has been revised adding new examples, explaining the difference between the exhaustive and the approximate methods and providing more details about the built-in MCS scoring functions.
- Figures have been added to *OEExprOpts Namespace* in order to demonstrate the effect of various atom and bond expression options on pattern matching.
- C++, Python, and Java manuals brought into closer alignment with each other.

5.83.2 OESystem 1.6.0

New features

- *OEInterface* allows multiple values per parameter. The values can be accessed by calling the *OEInterface.GetList* template member function of *OEInterface*.
- *OEAnnotation* class provides ability to attach various graphical objects (sphere, box, surface, etc) to classes derived from *OEBase* by using the generic data functions *OEBase.SetData* and *OEBase.GetData*.

Minor bug fixes

- The memory allocation performance of multi-threaded *OEChem* applications on both Windows and recent Linux/UNIX distributions (that use pthreads) has been dramatically improved. A new thread hashing algorithm is now used in *OESystem*'s memory pooling code which should dramatically reduce contention in allocation heavy multi-threaded code.
- A number of minor performance and numerical stability improvements have been made to *OEMath*'s geometry routines.
- When parsing the command line `--help -foo` is no longer sensitive to the case of `-foo`.

5.83.3 OEPlatform 1.6.0

Minor bug fixes

- The Microsoft Windows implementation of *OEMutex* has been rewritten to use interlocked intrinsics and semaphores instead of the Windows' `Mutex` objects, which were much slower.
- The *OETryMutex* class is now more efficient, using a platform-specific implementation on each supported target.

5.84 Version 1.5.1

5.84.1 OEChem 1.5.1

New features

- Added new *OEAssignFormalCharges* function that operates on a single atom instead of just a version for the entire molecule.
- Renamed the **OESystem::ParamVis** namespace to *OEParamVisibility* to make it consistent with other *OpenEye* namespaces.

Mayor bug fixes

- Fixed two bugs in kekulization of large molecules. First, some large molecules would fail kekulization when they were actually ok. Second, even when they were kekulized correctly, the method would still return false.
- This tweaks the MDL mol file reader to use the test `dimension != 3` instead of "dimension == 2" when deciding to honor the wedge/hash bonds or to determine the chirality from 3D coordinates. The subtle difference is that at the point this code is called, the dimension is not necessarily "2" or "3" if the (optional) header line is missing. If the header has been omitted, we treat the molfile like a 2D file (which it most probably is).
- Changed SMARTS parser to allow a TAB character (`\t`) to be treated as a separator following a SMARTS pattern. This reflects similar functionality in the SMILES parser and simplifies the task of writing "patty"-like applications.

5.85 Version 1.5.0

5.85.1 OEChem 1.5.0

OEChem 1.5.0 is a new release including many major and minor bug fixes along with several new features. This is also a continuation of a complete release of all *OpenEye* toolkits as a consolidated set so that there are no chances of incompatibilities between libraries.

Note, that in this release the directory structure has been changed to allow multiple versions of the toolkits to be installed in the same directory tree without conflicts. From this release on, all C++ releases will be under the openeye/toolkits main directory. There is then a directory specific to the version of the release and then below that, directories for each architecture/compiler combination. To simplify end user Makefiles, `openeye/toolkits/lib`, `openeye/toolkits/include'', and ``openeye/toolkits/examples` are all symlinks to the specific last version and architecture that was installed.

New users should look in `openeye/toolkits/examples` for all the examples. Existing users updating existing Makefiles should change their include directory from `openeye/include` to `openeye/toolkits/include`. As well, existing Makefiles should change the library directory from `openeye/lib` to `openeye/toolkits/lib`.

New features

- *OEChem* now has a 2D similarity implementation using the Lingos method of similarity. Lingos compares Isomeric SMILES strings instead of pre-computed fingerprints. This combination leads to very rapid 2D similarity calculation without any upfront cost to calculate fingerprints and without any storage requirements to store fingerprints.
- MMFF94 charges are now available in *OEChem*. While we recommend AM1-BCC charges as the best available charge model, having MMFF94 charges available at the *OEChem* level means that decent charges are available to all toolkit users.
- In *OEChem* 1.2, there was an alternate implementation of MCS that used a fast, approximate method for determining the MCS. While it is less than exhaustive, the speed does have some appealing uses. In *OEChem* 1.5, we've restored this older algorithm and now both are available.

```
namespace OEMCSType
{
  static const unsigned int Exhaustive  = 0;
  static const unsigned int Approximate = 1;
  static const unsigned int Default     = Exhaustive;
}
```

OEMCSType.Exhaustive implies the current, exhaustive algorithm from *OEChem* 1.3 and later, while *OEMCSType.Approximate* implements the older, fast but approximate algorithm.

- The ability to get the license expiration date when calling *OEChemIsLicensed* has been added.
- Molecules (*OEMol*, *OEGraphMol*) can now be attached to an existing *OEBase* as generic data and they will be written to OEB and read back in. Additional support for attaching grids and surfaces to molecules has been added to Grid and Spicoli.
- There is a new retain Isotope flag to *OESuppressHydrogens*. If false, `[2H]` and `[3H]` will also be removed by this call. By default, this is true so that the current behavior of *OESuppressHydrogens* is identical to the previous version.
- The *OEChem* CDX file reader can now Kekulize aromatic (single) bonds in the input ChemDraw file. It switches the internal bond order processing to use the bond's integer type field, and then calls *OEKekulize* to do all of the heavy lifting.
- Tweaks to the algorithm used for determining which bond(s) around a stereocenter should bear a wedge or a hash. The bug fixed here includes an example where all three neighbors are stereocenters, but two are in a ring and one isn't.
- There are new versions of *OEIsReadable* and *OEIsWriteable* that take a filename directly.
- More exceptional atom naming support for the PDB residues CO6 (pdb2ii5), SFC (pdb2gce), RFC (pdb2gce), MRR (pdb2gci), MRS (pdb2gd0), FSM (pdb2cgy) and YE1 (pdb2np9) has been added.

Major bug fixes

- The Bondi VdW radii tables have been completely updated and fixed. Additionally, any element not covered by the original Bondi paper is assigned a radius of 2.0 Ångströms.
- Corrected a serious glitch in the *OEChem* SMILES parser, where `[X-2]` would be incorrectly interpreted as `[X+2]`. This bug was an unintentional side-effect of the recent changes to warn about `[X+0]` and `[X-0]` when the parser is in strict mode.
- Fixed a bug in *OEKekulize* of very large molecules. Almost all small molecules continue to use the faster light-weight method, but we avoid exponential behavior for the large non-Kekulizable cases.
- Fixed a bug dealing with the semantics of wedge/hash bonds when the thin end is on a terminal atom. Clearly, in such cases the author probably intended to specify some aspect of the chirality of the atom at the other end of the bond. First, when drawing such structures into MDL's ISIS/Draw, it does not perceive specified chirality at the thick end of a wedge/hash bond from a hydrogen. However, this patch adds support for conveniently handling this case via *OEMDLHasIncorrectBondStereo* and *OEMDLCorrectBondStereo*, which notices this case, and either inverts the sense of the bond if the other end has explicit degree three or four, or removes the ignored *wedge/hash* bond annotation.
- Bug-fix improvement to *OEMDLCorrectBondStereo* such that whenever we introduce a new wedge/hash bond (on a previously unmarked bond), we now ensure that the we correctly set the thin end on the appropriate chiral atom.
- Fixed a bug in OEB reading/writing that makes sure that any unknown tags are simply passed from input to output without loss.
- Fixed a bug in reactions where the final valence was being computed incorrectly, such that the bond orders aren't modified directly until after valence correction is applied.

Minor bug fixes

- Tweaked the way the SMILES parser calculates implicit hydrogen counts on un-bracketed aromatic atoms in SMILES strings. This fixes `Scc` and `S1cccc1`.
- *OEParseCommandLine* now lists the legal and illegal values for a parameter if the user specifies an invalid value for the parameter.

- Fixed a bug where a warning was issued when writing >999 atoms to a V3000 MOL file. V3000 MOL files can contain >999 atoms, unlike V2000 files.
- Fixed a bug where round-off differences between Windows and Linux could result in stereo perception differences for 2D coordinates with angles very nearly equal to 180 degrees.
- Corrected the atomic number perception in the PDB file reader for the residues "G5P", "COK" and "COZ" (from the recently added PDB files pdb2f35, pdb2ges and pdb2g2z).
- Added residue perception support for the nonstandard amino acids "LYZ" and "MEN", which denote 5-hydroxylysine and N-methylasparagine.
- Fixed a bug where *OEGetHybridization* failed to classify the sulfur in C=S as sp2 hybridized.
- Completed the coverage of the MDL valence model to all atoms to handle charges less than or equal -2, or greater than or equal to +2. Many of the common cases of these strange charge states were already handled.
- The generic data reader now calls AddData instead of SetData. This means that multiple objects with the same tag can be handled appropriately.
- Added checks to prevent division-by-zero in several geometry routines in *OEMath*.
- *OESetComment* now safer to either a NULL pointer for the comment or a NULL pointer for the *OEAtomBase*.
- Fixed copy constructor and assignment operator for *OECoordArray*.

5.86 Version 1.4.2

5.86.1 OEChem 1.4.2

OEChem 1.4.2 is a bug fix release including dozens of minor and major bug fixes. It also includes numerous new minor features, and also brings significant practical changes. *OEChem* 1.4.2 is the first *OEChem* that is being released along with all the other public *OpenEye* toolkits. This will be a major step forward for inter-operability between different *OpenEye* toolkits and will open the way for easier support of customer applications that span multiple *OpenEye* toolkits.

OpenEye continues to be committed to maintaining a stable *OEChem* API and thus any programs written with previous versions of *OEChem* should re-build with the 1.4.2 version of *OEChem*. However, as is the nature of C/C++ code, particularly C++ template code, some of the bug fixes are in header files, so dependent libraries such as most other *OpenEye* libraries may require a re-link.

OEChem 1.4.2 will be the first release in which *OpenEye* is beginning to move toward synchronizing our library releases. This will remedy problems some people have experienced with link-problems when combining three or more *OpenEye* libraries into a single application.

Major bug fixes

- Added safeguards against stack over-runs in all molecule implementations.
- Fixed a major bug in rotor-offset compressed OEBinary file handling that made these molecules fragile with respect to molecule copies.
- Fixed thread-safety of reading SMILES.
- Fixed crash-bug when *OEPerceiveResidues* was called on unusual small-molecule peptide mimics such as O=CC(C)NC(C)C=O.
- Fixed dummy-atom handling in the CDX reader. Also improves implicit-hydrogen handling, but limits the reader to reading a single connection table.
- Corrected failure to include Isotope information in OEBinary files.
- Fixed crash-bug from processing stereochemistry on MDL atoms with degree three having hashes and wedges.

- A major oemolistream bug was fixed by fixing a stream bug (see *OEPlatform*'s Major bug fixes above).
- Fixed a bug where *OELibraryGen* failed to convert an aromatic ring to aliphatic accurately.

Minor bug fixes

- Dramatic improvement in the search speed of disconnected SMARTS such as `(*).(*)`.
- Speed optimizations were added to molecular geometry manipulations.
- Fixed very rare search-path bug in aromaticity perception.
- Allow support for `C[N@H]O` and `C[N@@H]O` in SMILES output.
- Fixed side-effects of *OECanonicalOrderBonds* on stereochemistry.
- Added BondIntType I/O in OEBinary format.
- Improve *OEKekulize* fall-backs to it fails gracefully.
- Add support for fully explicit Hydrogen flavor in SMILES (*i.e.* `[CH4]`).
- Fixed bug in the over-application of constraint for MCS searches.
- Added support for aromatic lead, aluminum, germanium and tin.
- Improved support for MDL's V3000 format.
- Improved perception of cationic N.pl3 atoms with partial charge greater than one.
- Added protection to SMILES writer for molecules with a terminal single bond that has stereochemistry specified.
- Fixed initialization of OEBinaryHandlers.
- Added support for the aromaticity model being used to effect chirality perception.
- Improved carbon dioxide, acetonitrile, cyanamide, cyanic acid and thiocyanic acid fragment recognition in bond perception algorithms.
- Fixed rounding error in *OENetCharge* function.
- Fixed endian bug in CDX reader for OS X.
- Added protection from molecule titles >4K in length.
- Added proper support for comments in MOPAC files.
- Improved copy construction and assignment of *OEQAtomBase* and *OEQBondBase*.
- Added *OEOFlavor.PDB.HETBONDS* flavor to the PDB writer that causes any bond to or from a `HETATOM` to be written explicitly. The HETBONDS flavor now part of the default output.
- Added support for `UNK` atoms attached to a backbone and `LIG` atoms that are covalently bound ligand or post-translational modifications to be given the same chain ID and residue number as the residue to which they are bonded.
- Added *OESymmetryNumber* to calculate the symmetry number of a given conformer within a user-defined threshold.
- Fixed bug that lost SD data attached to an *OEMol* (not its conformers) when the molecule was written to OEB (Note, SD data is not generally stored on the parent molecule).
- Fixed *OrderConfs* OS X memory bug.
- Fixed strip salts to only count heavy atoms in determining the fragment with the largest size. This makes the behavior remain consistent with implicit and explicit hydrogens.
- Added support for `x` in SMARTS as defined in v4.9 of Daylight's toolkit.

- Prevent PDB writer from generating `CONECT` records that have the same serial number for the source and destination.
- Fixed bug in automorphism calculations on molecules with hydrogen-defined cis-trans bond stereo.
- Improved algorithm for perception of 3D bond stereo.
- Added support for stable serial numbers, alternate locations, insertion codes and heteroatoms during calls to *OEPerceiveResidues*.
- Improved heuristics for residue perception in the presence of unusual chain terminations.

5.86.2 OEBio 1.4.2

Major bug fixes

- Changed OEHierView constructor to assume initial perception has already occurred. This prevents the constructor from over-aggressively re-perceiving residue information.

Minor bug fixes

- Made command-line usage more explicit for `subsetres` example.
- Improved output format at water categorization for `rescount` example.
- Converted *OEBio::GetChis* to *OEGetChis*.
- Improved support for g++ 4.x.
- Corrected bug so that a new chain automatically starts a new fragment and a new chain or new fragment automatically start a new residue.
- Fixed bug in **OEGetAtoms** from a residue's flood-fill algorithm.

5.86.3 OESystem 1.4.2

Major bug fixes

- Fixed bug that caused all the *OEErrorHandler* parameters to be reset to default values when the stream was reset.
- Increased error handler buffers to 8K to prevent crashes with very large warning and error messages.
- Added specific handling of *OEErrorLevel.Verbose* that was previously missing.
- Protect Hanoi sort routine from sorting lists of size 0 or 1.
- Prevent parameter mismatches in *OEMultiGrid*'s owned grids.

Minor bug fixes

- Improved efficiency of sorting in OEIter class and fixed bug in the *OEBinaryPredicate* sort implementation.
- Added Push, Sort and the NonConst constructor to the primitive specializations of OEIter.
- Allowed weighting of coordinates in inertial tensor and inertial alignment routines.
- Fix *OEBitVector.FromHexString* function to skip over end of line characters in the input.
- Allow *OEBitVector* to properly read `std::vector<bool>`.
- Allow quoting parameter values in *OEInterface* parameter files.
- Fixed bug in wildcard matching for *OEInterface* string parameters.
- Fixed auto-link warnings resulting from `--help html`.

- Added *OEFixedGrid.SetValues* (note, the size of the grid is not mutable).
- Fixed float/double precision bug in *OEStringToNumber*.

5.86.4 OEPlatform 1.4.2

Minor bug fixes

- Improved support for both MSVC and mingw on 32 bit Windows.
- Improved 64 bit support.
- Improved compatibility with HPUX and IRIX64
- Improved support for gcc 4.x.

Major bug fixes

- Fixed bug in *oeistream.close* that prevented *oemolistream.close* from returning the molstream to a default state.
- Fixed a size-check bug in low-level binary i/o routines.

5.87 Version 1.4.1

5.87.1 OEChem 1.4.1

OEChem 1.4.1 was an internal *OpenEye* release only. All associated release notes have been incorporated into the *OEChem* 1.4.2 release notes (see above).

5.88 Version 1.4.0

5.88.1 OEChem 1.4.0

OEChem 1.4.0 is a major new feature release. *OpenEye* is introducing *OEBio*, a new programming library extending *OEChem*'s convenience in handling biopolymers. In this initial release, *OEBio*'s API is small but useful. Over the life of the 1.4.x *OEChem* release series the *OEBio* API will grow. The purpose of *OEBio* is not to cover Bioinformatics, but to extend *OEChem*'s strong cheminformatics foundation to conveniently support protein modeling.

The source-code and examples in /openeye/examples/oechem have long been caught in a conflict. They served both as very useful tools and as didactic coding examples. To fulfill the role as tools, they needed good command-line-interfaces and error reporting. Unfortunately these features lead to more complex code. To fulfill a role as code examples, these programs need to be as simple as possible, highlighting one or two programming principles. In order to better serve both purposes, the example programs have now been split into /openeye/utilities and /openeye/examples, the first includes programs with more complex code and better interfaces and the latter with simple *OEChem* code examples. In addition, nine new example programs have been included to demonstrate common uses of the *OEBio* API.

In addition to *OEBio*, the 1.4.0 release includes many new features and bug fixes in the *OEChem*, *OESystem* and *OEPlatform* libraries.

New Features

- Split the programs previously in the examples directory into examples and utilities. The utilities directory will contain programs or versions of programs that may be useful and convenient for modelers to carry out common tasks. The examples directory will contain programs that may also be useful, but there primary purpose will be to provide didactic code examples of how to program common tasks using the *OEChem* library.
- New support for highly compact `rotor-offset compressed` oeb files.

- Added support for MDL ISIS Sketch file format with the `.skc` suffix.
- Added support for writing hydrogens that are required for specifying cis-trans stereo.
- Added support for `[Ds]` and `[Rg]` in SMILES and SMARTS.
- Added support for writing high-atomic number atoms in SMILES using `[\#123]` notation.
- New *OEWriteConstMolecule* function class to support high-level writing of const molecules. Introduced return-codes for the high-level writers that reflect that some molecules are inherently not supported by certain file formats (*e.g.* >999 atoms in .sdf).
- Add an *OEOFlavor.MOL2.Substructure* high-level writer flavor to force an `@TRIPOS<SUBSTRUCTURE>` idiom in the `.mol2` file.
- New *OEHasStereoHydrogens* function that determines if an atom has a proton that is required to specify stereochemistry.
- Added `retainStereo=false` default argument to *OESuppressHydrogens* that keeps hydrogens indicated by the *OEHasStereoHydrogens* function.
- Added *OEMatchBase.Clear* method.
- Dramatically improved efficiency of *DeleteConf* for deleting large numbers of conformers in order. Worst case behavior of the algorithm was changed from $N * N$ to N.
- Allow the SD file reader to handle a blank line between the `M END` and the '$ $ $ $' lines.
- Added convenience functions for getting and setting the MDL parity on atoms.
- Added new bitmask initialization parameters to *OEInitDefaultHandler* that allow easy specification of which handlers to initialize.
- New support for `h`, `d`, `t`, `[T]` and `[t]` nonstandard SMILES representations.
- Improved support for multiple NMR models in PDB files by reading, retaining and writing model number.
- Added fully supported **OEPDBData** and *OEPDBDataPair* classes as well as the necessary function to store and retrieve them from molecules.
- Three new convenience functions for clearing tag data: `OEChem::OEClearTagData`, *OEClearSDData* and *OEClearPDBData*.
- Added support for determining whether the library is properly licensed with *OEChemIsLicensed* function.
- Added *OEResidueHydrogens* function that will rename hydrogens an a heavy atom to their proper PDB atom names.

Major bug fixes

- Fixed *oemolistream.seek* and *oemolistream.tell* to take into account any cached molecules that may exist in the stream.
- Fixed low-level MDL reader to accept multiple SD tags with the same tag.

> **Note**
>
> It is not clear from the SD file specification if this is a valid SD format.

Minor bug fixes

- Added PDBData readers and writers to OEBinary file handlers.
- Added defensive code to *OEMolBase.DeleteAtom* and *OEMolBase.DeleteBond* to confirm that the atom or bond are owned by that specific molecule.
- Fixed rotation bug in inertial frame alignment.
- Converted inconsistent / and \ into a warning rather than an error, allowing the molecule to be parsed in a racemic fashion.
- Added an upper bound to the degree of the atoms at either end of a cis-trans chiral double bond.
- Added defensive code to prevent creation of atoms with atomic number greater than 255.
- Improved perception of non-aromatic exo double-bonds. This corrects a problem perceiving the progesterone in `pdb1a28`.
- Improved the exo-cyclic double bonds to sulfur. This improves the connectivity perception in `1hnv`, `1rev`, `1usn`, `1uwb`, `2usn`, `3usn` and `1zxv`.
- Improved the bond order perception of nitroso, oxime, azide, and arylhydroxylamine functional groups.
- Improved bond order perception of clashed structures by allowing hydrogens to only bond to their nearest heavy atoms.
- Prevent alternate conformation representations to be bonded to one another during bond perception.
- Made Up/Down choice for the first stereo bond in each resonance system canonical for writing isomeric smiles files.
- Made *OECanonicalOrderBonds* also order the bonds obtained with the *OEAtomBase.GetBonds* function call.
- Fixed bug in binary search for atomic number `0` used in *OEIsCommonIsotope*, *OEGetAverageWeight* and *OEGetIsotopicWeight*.
- Fixed the high-level pdb writer to preserve residue information found on the molecule.
- Corrected *OEIsReadable* to return false for the MOPAC file format.
- Added MOPAC flavors to the high-level molecule writers.
- Changed the hybridization assignment of negatively charged resonant nitrogens such as `*S(=O)(=O)[N-]C(=O)*`.
- Fix bug in *OESet3DHydrogenGeom* the could use a hydrogen's own coordinates as a reference for determining its geometry.
- Fix ring perception bug in *OEMCSMaxAtomsCompleteCycles*.
- Eliminate the redundancy between *OEChem::OEMDLSetBondStereo* and *OE3DToBondStereo* by allowing *OE3DToBondStereo* to take an optional bond mask and work on 2D as well as 3D molecules.
- Correct a bug in the *OEChem* interpretation of MDL wedge and hash bonds. In MDL connection tables, wedges and hashes only imply a specified stereo-center at the thin end (*i.e.* *OEBondBase.GetBgn*). This has been confirmed by comparing the wedge/hash bonds with the atom stereo parity bit in MDL ISIS output (including large vendor databases such as the entire Asinex 2005 collection).
- Fixed MDL reader bug where unrecognized atomic symbols would ignore subsequent fields in the atom block such as stereo parity, reaction role and valence.
- Added copy constructors and assignment operators to *OEMiniMols*, *OEChem::OEMiniBonds* and *OEMiniAtoms*.
- Fixed a sign error in *OESetAngle*.

- Added a length==0.0 check for *OESetDistance* and *OESetAngle*.

New Example Programs

These examples show the best feature of *OEChem*. Though most are less than 100 lines of simple code they demonstrate protein-protein sequence alignment, 2D and 3D structure manipulation, residue perception, robust multi-format I/O, stl integration, canonicalization, chirality perception and manipulation and many other complex cheminformatics tasks. While the main loop of each program is often only 30 lines long, it brings to bear thousand of lines of *OEChem* code and years of cumulative cheminformatics experience to easily combine 2D and 3D structure analysis and manipulation.

- `backbone.cpp`: Code to show the use of functors to select and write the backbone atoms of a protein.
- `cischeck.cpp`: Demonstrates how to loop over residues and checking the omega torsion for cis amides.
- `makealpha.cpp`: A code example of protein structure manipulation. This example modifies any protein into an alpha-helical structure with extended side-chains.
- `phipsi.cpp`: Simple code to report the phi-psi angles of a protein.
- `rescount.cpp`: Demonstrates an easy way to loop over the residues of a protein and query their information.
- `reshist.cpp`: Demonstrates and easy way to loop over a protein's residues and integrate the acquired data into an STL `dictionary` class.
- `seqalign.cpp`: This is perhaps the most complex program of the examples. It carries out protein-protein sequence alignment, alignment evaluation and printing as well as 3D structural alignment.
- `subsetres.cpp`: Simple code example of how to pull a specific residue out of a protein using its common name(*e.g.* ARG B 52).
- `swapaieres.cpp`: Demonstrates how a user can select a residue using its common name (*e.g.* GLN 252) and swap the ambiguous iso-electronic atoms.

5.88.2 OEBio 1.4.0

New Features

- Added *OESequenceAlignment* class with associated features for pairwise sequence alignment (including PAM250, BLOSUM62 and GONNET), writing an alignment to an *oeostream* and carrying out RMSD alignment between two proteins based on the sequence alignment.
- Simple methods for accessing and manipulating the torsion angles of biopolymers.
- Introduce classes that allow a hierarchical view of the Chains, Fragments and Residues of a protein while maintaining the efficient *OEChem* internal data structures.
- Added facility for swapping the terminal atoms of residues that are commonly ambiguous in protein crystal structures (*e.g.* terminal N,O of ASN).
- Added nine new example programs demonstrating the use of the new *OEBio* API points. These include: `backbone`, `cischeck`, `makealpha`, `phipsi`, `rescount`, `reshist`, `seqalign`, `subsetres` and `swapaieres`.

5.88.3 OESystem 1.4.0

New Features

- Moved superpose and tensor2mat API points from *OEChem* to *OESystem*. Added deprecated support for their use in *OEChem*.
- Added ability to assign an `OEIterBase<foo>*` to an `OEIter<const foo>` object. This allows much wider use of iterators of const objects.
- Made *OEIter.Sort* a stable sort.

- Additional physical constants added to *OEConst*.
- Added the ability to parse *OEInterface* parameter files without use of command-line parsing.
- New *OEChem::OEPDBOFlag::ELEMENT* and *OEChem::OEPDBOFlag::FORMALCHARGE* flavors for pdb writer. *OEChem::OEPDBOFlag::ELEMENT* adds the atomic symbol to columns 77-78 and *OEChem::OEPDBOFlag::FORMALCHARGE* add non-zero formal charges in columns 79-80.
- Extended the *OEOFlavor.SMI.ExtBonds* option from the `.smi` writer to the `.can` and `.ism` writers.

Major bug fixes

- Protected the *OEIter.Sort* function from NaN (not a number) members.

Minor bug fixes

- Fixed OEGrid and OEMultiGrid constructor bug that could cause no memory to be allocated for the grid elements.
- Corrected behavior of *OEGrid.Clear* to clear the *OEBase* data, remove the title and reinitialize all the elements of the grid.
- Fixed rotation bug in inertial-frame alignment.
- Fixed bug in the atom index into coordinates used while calculating the center of mass.
- Fixed bug in the calculation of *OEMultiGrid.SetSpacing* and *OEMultiGrid.SetMid* functions.
- Fixed *OEInterface* category name bug, `!KEYLESS` bug and unterminated category bug.

5.88.4 OEPlatform 1.4.0

New Features

- Improved binary data handling in streams.
- Significant improvements for user convenience in licensing code will allow future versions of *OpenEye* applications to manage licensing failures in a friendly manner.

Major bug fixes

- Fixed bug that prevented reading the final molecule in a file and then seeking to other positions in the file.
- Fixed a 64bit stream seek and read bug that could cause memory overflows and crashes.

Minor bug fixes

- Fixed bug in cross-platform directory searching and checking for files on a file system.
- Fixed bug in *OEPlatform::oeigzstream::size* that reported incorrect sizes in some instances.
- Added the ability to detect moved home directories under Windows.

5.89 Version 1.3.4

5.89.1 OEChem 1.3.4

New Features

- A new *OEChemIsLicensed* API point has been provided to allow an application to check whether the *OEChem*, *OESystem* and *OEPlatform* libraries have managed to find a suitable run-time license. This function can safely

be called from programs to determine *a priori* whether using *OEChem* functionality may trigger a fatal error, allowing a warning to be issued and/or calls the affected functionality to be avoided.

- A new `MOL2::Substructure` flavor has been added to the high-level writers (and a new default argument to the low-level *OEWriteMol2File*) to force the writer to emit a suitable `@TRIPOS<SUBSTRUCTURE>` record even for molecules having only a single substructure. The default behavior remains the same (to omit the record for *small* molecules) to save space in the common case.
- *OEChem*'s SMILES format parser, *OEParseSmiles*, has been tweaked to treat SMILES with inconsistent cis/trans bond stereo, such as `C/C=C(/C)/C`, as a soft warning rather than as a hard error. Previously, *OEChem* followed the Daylight toolkits, treating this as a serious error, and returning an empty molecule. The new behavior is that a warning is now thrown to *OEThrow*, but the molecule is returned, just without specified stereochemistry for the problematic bond(s). This allows *OEChem* to be used to correct/recover SMILES generated by buggy SMILES writers.
- The functions *OEAtomSetMDLParity* and *OEAtomGetMDLParity* have been introduced, to simplify the task of storing and retrieving the MDL parity value associated with each atom.
- The OEB file format readers and writers will now preserve PDB file header records, if they are present on the original molecule.
- A new function *OEGetCenterAndExtents* can be used to retrieve the center and 3D extents of a given *OEMolBase*.
- The file format extension `.isosmi` is now treated as a synonym for `.ism`, indicating isomeric SMILES.
- The *OEQMolBase.BuildExpressions* function now avoids constructing an atomic hybridization constraint if the query atom's *OEAtomBase.GetHyb* method returns zero.

Major bug fixes ——————————————————————————————^

- A problem with the generation of canonical isomeric SMILES has been fixed, that for pathological molecules with specified cis/trans stereochemistry could cause the canonical SMILES string to be written using `/` (forward slash) for some input orderings and `/` (backslash) for others. The resulting SMILES weren't incorrect (and the perceived symmetry invariants were fine), but this allowed equivalent isomers to be given non-identical isomeric SMILES strings.
- Fixed a bug in the 1.3.3 changes to the *OEPerceiveResidues* function. The new functionality to identify an acetyl as an N-terminal blocking group, PDB code `ACE`, could get confused when pattern matching some pathological graphs, such as proteins with N-terminal proline residues.
- Fixed a bug in the OEB file format writer that could result in the coordinates of a multiconformer molecule becoming scrambled if the atoms of the *OEMolBase* has been reordered, with *OEMolBase.OrderAtoms*, (and no atoms had been deleted) before writing the molecule. This could also affected the *OEMolBase.Compress* and *OEMolBase.UnCompress* methods.
- Fixed a bug that could cause a core dump when making a copy of an *OESubSearch* instance that contains atom or bond stereo.

Minor bug fixes ——————————————————————————————^

- Fixed a segmentation fault when calling either *OEIsCommonIsotope* or *OEGetIsotopicWeight* with both atomic number and atomic mass of zero. If ever *OEIsCommonIsotope* is called with a mass of zero (for any element) it now returns false, and whenever *OEGetIsotopicWeight* is called with a mass of zero, it now returns the corresponding isotopically averaged atomic weight, *i.e.* via *OEGetAverageWeight*.
- The *OEMatch* class was missing an explicit assignment operator, which could cause memory corruption if one *OEMatch* was ever assigned to another. This has now been implemented.
- Improved *OEChem*'s *OEDetermineConnectivity* functionality to restrict proximity-based bonding of hydrogen atoms preferably to their nearest suitable heavy (non-hydrogen) neighbor. Previously, hydrogens only bonded

to their nearest neighbor, which for all-atom molecules with bad clashes caused the overlapping protons to sublimate as molecular hydrogen, leaving their original parents in strange charge (or radical) states.

- The generation of `@TRIPOS<SUBSTRUCTURE>` records in Sybyl `.mol2` files has been rewritten to ensure that ligands, solvent molecules and nonstandard amino acids are correctly placed in their own Tripos substructure. Previously, PDB `HETATM` s would be placed in substructure 1, sharing it with a valid `RESIDUE`. Each *OEResidue* is now allocated a unique Tripos substructure, and nonstandard (or unrecognized) residues are marked as `GROUP`.
- The geometry functions *OESetAngle* and *OESetDistance* have both been made more robust to molecules without coordinates (and zero bond lengths).
- A refactoring of the SMILES generation routines in the *OEChem* source code means that the non-canonical SMILES created by the function *OECreateAbsSmiString* are now identical to those created by *OECreateSmiString* with just the *OESMILESFlag.AtomMaps* and *OESMILESFlag.RGroups* flags.
- Tweaks *OEDetermineConnectivity* to avoid generating bonds between atoms in alternate locations/conformations. This is only a problem for PDB files read with the `ALL` input flavor. There are still multiple bonds between invariant *hinge* atoms and the multiple copies of its neighbors in alternate conformations.
- Fixed a potential performance problem in *OEMolBase*'s *OEChem::OEMolBase::operator+=*, and the equivalent *OEAddMols* function calls, when concatenating large molecules with coordinates.
- The performance of reading MDL SD files containing a large quantity of SD tag data and for PDB files containing a large amount of header information (with the `DATA` flavor) has been improved. Instead of attaching each data item to the molecule as it's encountered, the file format readers now accumulate the data first, and then attach it to the molecule on encountering the end of the connection table.
- The performance of handling generic data tags when reading OEB files containing large numbers of atoms or bonds has been improved.
- The consistency of file format flavors have been improved. The SMILES *OEOFlavor.SMI.ExtBonds* flavor that was previously only available to the *OEFormat.SMI* file format, is now also available to the *OEFormat.CAN* and *OEFormat.ISM* formats. Likewise, the flavors available to the *OEFormat.MOL2* and *OEFormat.MOL2H* file formats have been synchronized, with *OEFormat.MOL2H* becoming a flavor of *OEFormat.MOL2* (*i.e.* the file extension effectively specifies a different default flavor).
- The *OEMCSMaxAtomsCompleteCycles* objective function requires that ring atoms and bonds of the query molecule have been perceived (using *OEFindRingAtomsAndBonds*) before the corresponding *OEMCSSearch* instance is constructed. To avoid unexpected behavior, the *OEMCSSearch* constructors now call *OEFindRingAtomsAndBonds* themselves if allowed to make a copy of the query molecule (the default), or issue a warning to *OEThrow* otherwise.

5.89.2 OESystem 1.3.4

New Features ——————————————————————————^

- New *OEMath* template functions *OEGeom3DGetCenterAndExtents* and *OEGeom3DGetCenterOfMass* have been added to calculate the center, extents and (weighted) center of mass of a set of 3D coordinates.
- The *OEInterface* class now has new *OEInterface.GetTypedParameter* template member functions.
- A new function *OEGetCenterAndExtents* has been added to *OESystem*'s grid handling code, to retrieve the center and extents of a grid.

Minor bug fixes ——————————————————————————^

- A complete new implementation of *OEConcatIter* to avoid potential problems caused by the fact that appending items to an STL `std::vector` invalidates iterators over it. The new implementation uses offsets rather than pointers to avoid problems. This also fixes a crash of *OEConcatIter.operator++* on an empty iterator.
- Fixed a bug in *OEIter.Push* that caused problems if the iterator was previously empty.

- Fixed a bug in *OEFileExtension* that was causing crashes when no extension was present.
- Fixed a bug in *OEInterface.GetParameters* that caused parameters in sub sub interfaces to appear multiple times in the returned iterator.
- The *OEWriteSettings* function now takes a *const OEInterface* reference.

5.89.3 OEPlatform 1.3.4

New Features ——————————————————————————^

- New functions *OEFileDetermineName* and *OEFileDeterminePath* to extract the basename and path of a filename respectively.

Minor bug fixes ——————————————————————————^

- Fixed a problem in the *oeigzstream* constructor, that could cause problems determining the size of a file using `size`, as zlib was not being correctly initialized.
- Fixed a minor bug in *OEFileCreate* that was failing to close the file descriptor for the newly created file.
- The return type of the *OEFile.Size* method and the *OEFileSize* function have been changed from `size_t` to `oefpos_t` to allow correct handling of large files on 32-bit platforms.
- Fixed some minor bugs in *oeistream* and *oeostream* when these classes are used directly.

5.90 Version 1.3.3

5.90.1 OEChem 1.3.3

New Features

- Several enhancements have been made to the protein perception algorithms used in *OEPerceiveResidues*. These allow *OEChem* to recognize the N-terminal capping group `ACE`, and the nonstandard amino acid residues `ABA`, `CGU`, `CME`, `CSD`, `MLY`, `MSE`, `PCA`, `PTR`, `SEP` and `TPO`. Support for these additional amino acid types has also been added to *OEGetResidueIndex* and friends. The sidechain pattern matching algorithm now has improved *fallback* functionality for better handling of modified/substituted residues.
- Improved support from aromatic boron and aromatic silicon in *OEKekulize*. The *OEChem* toolkit currently doesn't perceive either boron or silicon to be aromatic (with any aromaticity model), but this enhancement allows us to Kekulize structures so specified.
- Added improved support of parsing SMILES containing aromatic boron and aromatic silicon, allowing the *OEChem* toolkit to parse `b1ccccc1` (borinine).
- A new *OEGetDelphiRadius* function has been added to *OEChem* to return the default radius for a given element used by the Accelrys' Delphi program for electrostatics calculations.
- A new function *OEGetAminoAcidCode* can be used to convert an index from the *OEResidueIndex* namespace to a IUMB single character code (`A` for alanine, `R` for arginine, etc…).
- Several new convenience functions, *OEAssignCovalentRadii*, *OEAssignDelphiRadii*, *OEAssignBondiVdWRadii*, *OEAssignPaulingVdWRadii* and *OEAssignHonigIonicCavityRadii*, are now provided to set the radius property on each atom of a molecule to the value specified by the corresponding `OEGet...Radius` function.
- A new function *OEIsBinary* is provided to determine whether the specified file format is binary or not, for example, `.oeb`, `.bin` and `.cdx`.
- The new function *OEGetFormatExtension* can be used to return a comma separated list of lowercase file format extensions that can be used to aid implementing directory scans and file format dialog boxes.

- A new *OEMCSFunc* functor, *OEMCSMaxBondsCompleteCycles* can be used as an objective function to *OEChem*'s maximal common subgraph matching algorithms.

Major bug fixes

- A problem in *OEChem*'s graph canonicalization algorithm was identified by the NCBI's PubChem project for the single molecule: `C12C3C4C3C5C4C1C25`. This problem has been fixed in *OEChem* 1.3.3. Unfortunately, this failure didn't show up on our testing of 100 random permutations of 2.5 million compound test set. Efforts are now on-going to validate *OpenEye*'s canonicalization against all theoretical connection tables with less than N atoms, for some $N > 10$.
- A bug in the OEB file format readers and writers that could cause the titles and/or comments attached to molecules or conformers to be lost, has been corrected.

Minor bug fixes

- Fixed bug in the *OEChem* SMARTS parser that failed to follow the Daylight semantics for patterns such as `[H]`, `[2H]` and `[H+]` where the `H` specifies the pattern must match a hydrogen, and not the expected hydrogen count on an atom.
- The *OEChem* SMILES writers have been modified to prevent them generating atoms such as `[C@H2]` or `[C@@H2]` for centers that have stereo explicitly specified (on non-chiral centers) with explicit hydrogens, when the hydrogens are being automatically suppressed by the output SMILES flavor.
- The methods *OEAtomBase.SetStereo*, *OEAtomBase.GetStereo*, *OEBondBase.SetStereo* and *OEBondBase.GetStereo* have been enhanced such that the internal representation of stereochemistry is invariant of hydrogen suppression. The functions *OESuppressHydrogens* and *OEAddExplicitHydrogens* no longer invalidate stereochemistry.
- The old-style OE binary, `.bin`, file format reader now automatically sets the dimension property of molecules and conformers to 3. Whilst new-style OE binary, `.oeb`, files explicitly record the dimensionality of the stored coordinates, the old format didn't and its contents should be assumed to be 3-dimensional.
- Correct a minor logic problem in *OEQMolBase.BuildExpressions* when constructing the expressions to match bond orders but not aromaticity.
- Fixed a problem in the SMILES parser, which would cause a segmentation fault if ever a SMILES string longer than 4096 characters encountered a syntax or Kekulization error. We no longer try to report the location of the syntax error for SMILES strings longer than 2048 characters.
- A bug in *OEPerceiveBondOrders* that assumed/required that the incoming molecule not have any aromaticity specified, has been fixed by calling *OEClearAromaticFlags* on the incoming molecule. This assumption was valid for its existing use by the high-level file format readers, but meant that calling *OEPerceiveBondOrders* twice in a row could sometimes produce different results.
- Fixed a potential problem in several file format readers that caused a run-time abort in Microsoft's runtime libraries on Windows when reading corrupt or binary files. The Microsoft implementation of the standard `<ctype.h>` functions, such as `isdigit` and `isupper` will abort when passed negative values, such as when interpreting the bytes of a file as (signed) char.
- Fixed a segmentation fault in *OEScrambleMolecule* that was triggered by chiral molecules.
- Fixed a bug in *OEMDLCorrectBondStereo* that could cause that routine to crash, if the chiral atom on which the stereo chemistry needed to be corrected was degree three instead of degree 4. This routine has been made more robust, and can now correct wedges and hashes around degree three atoms that conflict with the specified MDL parity bit.
- The *OEChem* MDL mol file reader has been made more robust by checking for negative values in the atom count, bond count and list count fields. These are now interpreted as being zero. Corrupted SD files could previously cause *OEChem* to crash.

- Calling close on an *oemolistream* that wraps `OEPlatform.oein`, will now correctly make `oemolistream.operator bool` return false, and stop it reading (even though `OEPlatform.oein`, itself shouldn't be closed).
- The *OEChem* SMILES parser, `OEParseSmiles` function, has been fixed to set the default bond order of unspecified external bonds, *i.e.* `C&1`, to be single. Previously these were left initialized as bond order zero, although `C&=1` and `C&#1` were correctly handled as double and triple bonds respectively.
- The function `OEPDBOrderAtoms` has been improved to only compare atoms names for recognized residues when sorting. This prevents atoms being needlessly reordered for no good reason.
- `OEPerceiveResidues` has been improved to assign unique atom names to every atom within an unknown or unrecognized residue. Previously, all six atoms in benzene would be given the same atom name `C `` which confuses software that assumes PDB atom names are unique within a residue. *OEChem* now assigns ``C1`, `C2`, etc...
- Add goof-proofing to return calls to `OEInvertCenter` where the specified atom is not trivially invertible (*i.e.* a center with 3 or more ring bonds).
- Improved handling of the hydrogen isotopes `D` and `T` when reading MDL connection tables. These symbols now automatically set the isotope field appropriately. Previous versions of *OEChem* interpreted these symbols as forms of hydrogen, but relied on the MDL's mass field or `M ISO` line being correctly set to specify a/which isotope.
- A very minor bug in `OEPerceiveResidues` has been fixed that prevented residue information from being assigned to lone protons. The algorithm previously assumed all hydrogens were bonded to a heavy atom parent.
- In `OESubsetMol` the dummy atoms used to represent attachment points are no assigned map indices starting from one, *i.e.* R1, R2, R3, instead of from zero, *i.e.*, R1, R2.
- `OESubsetMol` now attempts to preserve or undefine the specified stereochemistry at atoms and bonds affected by attachment points.
- The performance of `OEDetermineConnectivity` has been dramatically improved for very large molecules. This greatly speeds up the reading of proteins like `pdb1jj2.ent` (which contains 98,543 atoms) several fold.
- Replaced an inefficient $O(n^2)$ algorithm in the *OEChem::OEMolBaseImpl::OrderAtoms* method that checked that the input vector was a valid permutation of a subset of the atoms in the molecule. This dramatically improves the performance of writing large PDB files.
- The performance of many of the *OEMolBase*, *OEAtomBase* and *OEBondBase* methods has been improved in *OEChem* 1.3.3.
- The methods `oemolistream.operator bool`, `oemolostream.operator bool` and `oemolistream.eof` have been marked *const* to enable better compiler optimization.

Java wrappers

New Features

- With this release of *OEChem*, Java wrappers are now provided. This first version only supports Sun's JVM version 1.4.2.

Python wrappers

New Features

- The *OEInterface* class and associated machinery for creating and parsing command lines is now available in Python. While Python has native command line argument support, this provides an alternative that is functionally similar to the C++ *OEChem* version. The example program `molextract.py` has been updated to demonstrate this new feature.

Major bug fixes

- Fixed a memory leak in *OENot*, *OEAnd* and *OEOr* predicates.
- Fixed a bug in **PyAtomPredicate**, **PyBondPredicate** and **PyConfPredicate** where a syntax error in the Python callable function would silently fail. Now, if there is an error in the Python function, the exception will propagate back to the Python interpreter.

5.90.2 OESystem 1.3.3

New Features

- By default the *OpenEye* toolkits now use thread-safe memory management internally to allow multiple molecules (and other objects) to be manipulated by different concurrent threads. Modifying the same object concurrently is still unsafe. On some operating systems, *OEChem* intensive applications may experience a slight overhead which may be explicitly disabled with the new `OESetThreadSafe` function call. Timings on modern GNU/Linux systems show almost no overhead, and the performance benefits of upgrading to g++ 3.4.x means that most applications should run faster with *OEChem* 1.3.3 than with previous releases even with thread-safety enabled.
- The `--help` functionality of the *OEInterface* class has been improved to indent and wrap the on-line help text at 80 columns. The default screen width can be controlled by specifying the column width on the command line, for example `--help all 100`.
- The *OEInterface* parser has been improved to allow `!CATEGORY` names to be quoted, allowing names to contain spaces.
- The *OESystem::OEFizzGrid* class now has an *OESystem::OEFizzGrid::operator bool* method, which returns true if either floats or integers have been set.

Major bug fixes

- The semantics of how quaternions are represented within the *OpenEye* toolkits have now been standardized, as scalar-first. Hence, of the four floating point values that define a quaternion, the first represents the scalar component and the final three values represent the vector component. The failure to explicitly document which of the two possible forms was used, resulted in some *OEMath* functions assuming scalar-first whilst others assumed scalar-last. (The quaternion functions in OELib, for example, used scalar-last). Functions affected by this include *OEMath::OEGeomQuaternionMultiply*, *OEMath::OEGeom3DUnitQuaternionRotate* *OEMath::OEGeom3DQuaternionToRotMatrix* and *OEMath::OEGeom3DRotMatrixToQuaternion*.

Minor bug fixes

- Fixed a potential memory leak in *OEBinaryNot*.
- *OEInterface*'s methods `OEInterface.DeleteInterface` and `OEInterface.DeleteParameter` now recursively search through sub-interfaces for the object to delete.
- The `OEStringTokenize` and `OEStringTokenizeQuoted` functions have been completely rewritten. Both previous implementations could potentially thrown C++ exceptions, and the latter was just plain broken.
- A minor bug in the *OEInterface* class, that in some cases caused the detailed description to end with `!END`, has been fixed.
- The behavior of the `!REQUIRED` keyword has changed in *OEInterface* files. If an option has a default value, specified by the `!DEFAULT` keyword, then the `!REQUIRED` option is ignored.

5.90.3 OEPlatform 1.3.3

Minor bug fixes

- Modified *OEFileDeterminePathAndName* to canonicalize directory separators to the appropriate form for the host operating system.
- Improved the performance of *OEMutex* when using g++, by using the low-level `gthr` API, rather than using the higher-level locking primitives used by the `libstdc++` STL library.
- Fixes to *oestream* classes to prevent accidentally closing `stdin`. Minor bug fix to *oeiwrapperstream* implementation.

5.91 Version 1.3.2

5.91.1 OEChem 1.3.2

New Features

- New function *OEReadRxnFile* to read MDL RXN (reaction) file format.
- New functions *OEGetAtomComment* and *OESetAtomComment* that allow arbitrary text strings to be associated with atoms. This is currently used to preserve/manipulate atom aliases from MDL file formats.
- Changes to *OEReadMDLFile* and *OEWriteMDLFile* to read and write atom alias information. Atom alias information is stored on an atom by the reader using *OESetAtomComment*, and any such atom comments are written to the MDL connection table on output.
- New function *OEInvertCenter* to invert a tetrahedral center. For chiral atoms, this function flips the molecule from one isomeric form to the other. This function returns false if the center can't easily be inverted.
- Query molecules, represented by an *OEQMolBase* can now be read from and written to OEB binary files.
- New function(s) *OECorrectAcidProtonGeometry* to expose the new functionality in *OESet3DHydrogenGeom* to ensure that protons on carboxylic acid groups are cis.
- A new SMILES flavor, *OESMILESFlag.ExtBonds* allows *OEChem* to write out SMILES strings using the external bond notation, {*i.e.* `[*:1]CC[*:2]` (a.k.a. `[R1]CC[R2]`) can now be written out as `C&1C&2`.
- Constructors that take an STL `std::string` have been added to both *oemolistream* and *oemolostream*, allowing then to take C++ strings, in addition to `const char*`.
- A *oemolistream.rewind* method has been added to *oemolistream* to match the functionality available with *oeistream*. This method rewinds the stream to the beginning (if possible) and is equivalent to *oemolistream.seek* (0).
- Added a new *OECliqueSearch.SetMCSFunc* method to *OECliqueSearch* to allow callers to customize the MCS function used in clique searching.
- A new constructor to the *OEIsMember* functor allows it to be used with an STL *std::set*. Additionally *OEIsMember* and *OEIsMemberPtr* now have assignment operator *OEIsMemberPtr.operator=* and *OEIsMemberPtr.Set* methods.
- Added new function *OEGetHonigIonicCavityRadius* to return the effective ionic radius of each element to be used in solvation calculations. These values are described in [Rashin-1985] .
- New example programs `catmols`, `movemol`, `pdbdata`, `rings`, and `rmsd_selftest`.

Minor bug fixes

- Fixed a problem with reading multi-conformer molecules from MDL SD file format, where we wouldn't correctly record that each conformer already had MDL atom parity information (so we'd re-perceive it upon output which wouldn't preserve the original input values).
- The dimension of an *OEMolBase* (as returned by the *OEMolBase.GetDimension* method), is now read and written to OEB binary files. Previously, all molecules in OEB were implicitly 3D (which remains the default), but we now explicitly record when the dimension has a value other than three.
- Fixed a bug in *OEPerceiveChiral* that failed to recognize that double bonds in rings of size seven or greater are potentially chiral.
- Fixed an obscure bug in the MDL file reader. We were doing chirality and bond stereochemistry processing before we'd kekulized any aromatic bonds (from MDL substructure queries) or set the implicit hydrogen counts. This occasionally confused *OEPerceiveSymmetry*'s graph invariants and *OEPerceiveChiral*'s tests for potential chiral atoms.
- Corrected the high-level file writer, *OEWriteMolecule*, such that when calculating MDL atom parity bits prior to writing MDL file formats, to first extract chirality from 3D coordinates if present. This is now consistent with similar logic prior to writing isomeric SMILES.
- When writing isomeric SMILES, the *OECreateSmiString* function was suppressing explicit hydrogens necessary to represent double bond stereochemistry. These necessary *stereo* hydrogens are now written out explicitly.
- Fixed some issues in the *OEChem* SMILES parser where there was insufficient validity checking in the processing of external bonds. These problems caused SMILES strings such as `C&1&1`, `C&1C&1` and even `&=9` to crash the parser. We now more politely generate a warning message and return false.
- Improvements to *OESet3DHydrogenGeom* to improve the geometry of protons added to carboxylic acids (which are now guaranteed to be cis). *OESet3DHydrogenGeom* also avoids calling the function *OEAssignHybridization* on the molecule, using a user-assigned hybridization if available and calling *OEGetHybridization* otherwise.
- Several fixes and improvements to the SMARTS parser. We weren't correctly handling SMARTS with double digit ring closures, and we didn't recognize `[te]` as aromatic tellurium. Also fixes to cis/trans stereo chemistry in a few corner cases.
- Fixed by in a bug in reaction handling of the *OELibraryGen* class that caused a segmentation fault when using a reaction to delete unmapped atoms.
- Fixed a bug in *OERMSD* where if asked to overlay two sets of coordinates, and the user didn't ask for either the rotation matrix or translation vector, and the two structures overlaid perfectly, we'd generate a segmentation fault (we were writing a unit matrix to a NULL pointer).
- Fixed a bug in *SweepConfs* where we'd fail to renumber conformer indices, if there were no deleted conformers.
- Minor tweak to *OEGetHybridization* for uncharged sulfur, selenium, tellurium and polonium. These are now always considered sp3 unless they're aromatic, in which case they are sp2.
- The *OEQBase.SetExpr* has been changed to make a copy of the passed *OEExprBase* pointer which fixes issues with copying/assigning *OEQMolBase* s.
- Fixed a rare bug in *OEAtomBase.GetStereo* that could occasionally return either *OEAtomStereo.Left* or *OEAtomStereo.Right* for an atom without *OEAtomBase.HasStereoSpecified*. This has now been fixed such that whenever *OEAtomBase.HasStereoSpecified* returns false, *OEAtomBase.GetStereo* will always return *OEAtomStereo.Undefined*.
- A minor tweak to *OEGetFileType* to perform case-insensitive string comparison, allows this code to recognize file format extensions independent of capitalization, *i.e.* `.mol2` and `.MOL2`.

- Improvements to the heuristics used by *OEMDLPerceiveBondStereo* for placing wedge and hash bonds on a connection table. These include avoiding placing the wedge/hash on the fusion bond for chiral bridgehead atoms, and arbitrarily choosing amongst the best bonds when no unique best is found (previously a tie caused the algorithm to choose randomly between all neighboring bonds).
- Tweak to *OEDetermineConnectivity* such that atoms that are marked as PDB residues `CL`, `BR` and `IOD` (*i.e.* chloride, bromide and iodide ions) are treated like solvent, and are never bonded to other residues (in this case other atoms).
- Routine maintenance improvements to *OEChem*'s PDB file format readers to handle files recently released by RCSB. For example, residue `GCP` in pdb1rca.ent, residue `783` in pdb1o2t, and similar fixes in PDB codes 1ta2, 1ta6, 1o3m, 1o3n and 1o3o.
- Improvement to *OEReadMacroModelFile* to read the atomic partial charge out of the charge-charge interaction column and fixed a bug in extracting the atom name field.
- Enhancement to the *OEWriteMacroModelFile* function to write out the atomic partial charge to both the charge-charge and charge-multipole fields of the MacroModel connection table. Previously, we only stored the partial charge in the charge-charge field, and wrote 0.0 to the charge-multipole field.

Python wrappers

Major bug fixes

- Fixed a major bug in SD tag handling for *OEGraphMol* s. In *OEChem* 1.3.1, re-using an *OEGraphMol* and reading from an SD file, could result in one molecule getting SD tags from the previous molecule.

Minor bug fixes

- Alternate implementations of *OEGraphMol* are now exposed by passing a constant to the constructor. This was exposed in 1.3.1, but the C++ constants from `molfactory.h` were not wrapped.
- molchunk.py stopped working in 1.3.1 due to changes to *oemolistream.seek* on a stream. A new method *oemolistream.rewind* has been added to *oemolistream* to take the place of the old seek(0).

5.91.2 OESystem 1.3.2

New Features

- A new two argument variant of *OEGeom3DAngle* allows the calculation of the angle between two vectors. This is an efficient form of the three argument variant, where the middle point is defined to be the origin.
- Added a new *OELinearInterpolate* function (template) that can be used to linearly interpolate the value at an arbitrary point inside an *OEFixedGrid*.
- Two new functors *OEBinaryAnd* and *OEBinaryOr* for constructing a single binary predicate from two binary predicates. These are equivalent to the *OEAnd* and *OEOr* functors for unary predicates.

Major bug fixes

- Exposed the `_baseimpl` member of *OEBase*. This allows the *OEChem* python wrappers to associate generic data correctly with a molecule.

Minor bug fixes

- Made *OEBitVector*'s destructor virtual to allow classes to be derived from them.

5.91.3 OEPlatform 1.3.2

Minor bug fixes

- Fixed the STL standard file I/O wrappers, *oeistdstream* and *oeostdstream* that wrap/transform C++'s `std::istream` and `std:ostream` into the *oeistream* and *oeostream* required by *OEChem*'s file functions.

5.92 Version 1.3.1

5.92.1 OEChem 1.3.1

Major bug fixes

- Fix cis/trans bond stereochemistry perception when reading MDL MOL and SD files via *OEReadMDLFile*. We were overly aggressive when marking double bonds in rings as cis vs. trans due to a missing call to *OEFindRingAtomsAndBonds*. We also no longer attempt to attribute E/Z stereochemistry to connection tables without coordinates, or when any of the relevant bond lengths are zero.

Minor bug fixes

- Accidentally *OEChem*'s zmatrix handling functions were documented in the *OEChem* documentation but were missing from the distributed list of header files. This oversight has been corrected and the classes *OEChem::OECartesianToInternal* and *OEChem::OEInternalToCartesian* and the functions *OECalcCartesianCoord* and *OECalcInternalCoord* are now publicly available.
- Handle (and optimize) the case where *OECanonicalOrderAtoms* or *OECanonicalOrderBonds* is called with less then two atoms.
- Fixes *OEGetSmallestSubtree* to correctly return the smallest set of atoms on either side of a given bond when the molecule contains disconnected components.
- Performance improvements to **OEGraphMolParameter**, to avoid using *OEIsomericConfTest* when loading single conformer molecules.
- Bug fix to *DeleteConf* that could occasionally result in that multi-conformer molecule's active conformer being corrupted.
- Tweak to *OEMDLPerceiveBondStereo* to preferentially place the wedge or hash bond on the bond to an explicit hydrogen, for chiral stereo centers of degree three.
- Enhancements to *OEMDLPerceiveParity* to support chiral atoms with two heavy atom neighbors and an explicit hydrogen. These aren't supported by MDL software (including ISIS/Draw) but this allows *OEChem* to convert `X[N@H]Y` to an MDL mol file and back to isomeric SMILES without loss of information.
- Improvements to *OEMDLStereoFromParity* to set *OEChem*'s atomic chirality for the MDL stereo parity flag for both degree three and degree four atom stereo centers.
- Correct problem in *OEChem::OEMiniAtom::Copy* where duplicating an atom would preserve the explicit degree of the original, as returned by *OEAtomBase.GetExplicitDegree*.
- Fixes to both *OEChem::OEMiniBond::SetOrder* and *OEChem::OEMiniBond::SetIntType* that would incorrectly reset the aromaticity and in-ring flags of a bond as a side-effect.
- Miscellaneous fixes and improvements to *OEChem::OEMiniMol::Copy*.
- Fix to *oemolostream.openstring* to allow an *oemolostream* to be reused multiple times, rather than constructed and destructed for each use.
- *OEPerceiveSymmetry* would previously segmentation fault on some platforms if passed a molecule with no atoms. This routine has now been idiot-proofed.

- Maintenance updates to *OEReadPDBFile* to handle PDB files deposited at RCSB, up to July 2004. For example, correctly handling the hetero atoms of the NZQ residue of pdb1oj7.ent.
- A serious performance regression in *OERMSD* has been addressed. A change to *OERMSD* for *OEChem* 1.3 unexpectedly resulted in a significant drop in superposition speed. This has now been resolved and *OERMSD* is as fast as (or faster than) it was in *OEChem* 1.2.
- The string reference argument to *OESetComment* is now correctly marked as *const*. This value is never modified by this function.
- The sanity checking in *OESetTorsion* has been improved such that we now return false for dubious calls to that function. For example, when the first and four atom pointers refer to the same atom. Previously, we'd do nothing, but not indicate a failure with the return value.
- Improvements to match limit test in *OESubSearch* that prevent excessive run-times for pathological substructure search patterns. Previously, substructure searching of `*.*.*.*` wouldn't honor the match limit setting, and spew multiple *match limit reached* warnings.
- Enhancement to *OEWritePDBFile* to allow it to honor the *OEOFlavor.PDB.RADIUS* flag to write an atom's radius in the PDB occupancy field, even when writing a molecule without residue information.
- Better handling of out of range coordinates in *OEWriteMDLFile*, *OEWritePDBFile* and *OEWriteMacroModelFile*. Previously, a Cartesian coordinate larger than these file format's fixed width fields would corrupt the following fields on the same line.

5.92.2 OESystem 1.3.1

Minor bug fixes

- A fix to the function *OECheckHelp* that prevents displaying on-line help for HIDDEN command line interface options.
- Fixed a bug in *OEInterface.GetInterface* that would fail to find the correct interface by name.
- Correct the return value for *OEInterface.DeleteInterface*. We now return false if we failed to delete the given interface.
- Fixed an off-by-one error with the *OEPredVectorPtrIter::ToLast* method on some kinds of iterators (specifically, *OESystem::OEPredVectorPtrIter* s).

5.93 Version 1.3.0

5.93.1 OEChem 1.3.0

New Features

- Added reaction parsing and processing including: *OELibraryGen* class for generating combinatorial libraries; *OEUniMolecularRxn* class primarily for applying normalizations; *OEParseSmirks* function for converting Daylight's SMIRKS format to an *OEQMolBase*; and *OEReadRxnFile* that takes an *OEMolBase* for reading MDL's rxn files.
- Added *OEVectorBindings* class to handle SMARTS vector bindings.
- Added ChemDraw CDX reader and writer functions.
- Added ability to perceive symmetry classes useful for automorphism detection.
- Added factory methods for allocating molecules. Implemented *OEGraphMol*, *OEMol* and *OEQMol* in terms of the factory methods.
- Added `int` and `float` grid handlers to *OEBinaryIOHandler*.

- Added *OEDisassembleExpressions* function which converts query expressions into data directly on atoms/bonds.
- Added OEMiniMol – a small memory footprint molecule useful for in-memory graph searching.
- Added utility functors *OEUnaryToBinaryAnd* and *OEUnaryToBinaryOr* that turn a unary functor into a binary 'or' or 'and' functor.
- Add default 64 bit file support on Linux.
- Added functions to set and get comment on molecules. These comments are written to the appropriate fields of .sdf, `.mol2` and `.oeb` file formats.
- New function *OEMDLStereoFromParity* that sets the *OEChem* stereochemistry of each atom from the MDL stereo parity flag stored in generic data.

Improvements and Optimizations

- Performance enhancement to *OESetDistance*.
- Modified the implementation of the *OEMCMolBase* member function *SetActive* for significant performance gains in binary i/o.
- Optimization of copy construction of *OEMCMolBase*.
- Added warning when *oemolistream* and *oemolostream* fail to allocate a *oeogzstream* or *oeigzstream* .
- Made *OEMol* and *OEMCMolBase* contain a separate dimensionality (*e.g.* - the value returned by *OEMolBase.GetDimension*) for each conformer rather than a single value for the molecule. This allows depictions and conformers to be stored on the same molecules and still be handled correctly by the .sd writers.
- Numerous improvements/enhancements to *OEChem*'s PDB file reader. The list of exceptional residue names has been updated to match the May 2004 list of public PDB files from RCSB.
- The MDL mol file reader is tweaked to record unrecognized atomic symbols (just like we do for 3-letter superatom codes) storing them in the atom's name property/field.
- The routine *OEMDLPerceiveParity* (and thereby the MDL mol file writer) is improved to record chiral centers that don't have stereo specified as the MDL parity value 3 (unspecified chiral center).
- Improved performance for *OEMolBase* member functions *OEMolBase.GetAtom*, *OEMolBase.GetBond* and *OEMolBase.Sweep*.
- All parity values were being stored for atoms (including value == 0) which was unnecessary, and was consuming both memory and space on disk in the form of OEB files.
- Added call to seed random number generator from the clock in *OERandomizeTorsions*.
- Added more `float` multi-grid handlers to be initialized by default.
- A warning message was added to the SMARTS parser to alert users in cases of attempted assignment of map index to zero.
- Moved implementation of *OEInSamePart* to a template class so that parts can be tested for bonds (and other objects) as well atoms.
- Extended to *OEIsRotor* constructor to allow the option of hydrogen rotors.
- Created and exposed *OEMDLStereoFromBondStereo* function from previously internal code.

Major bug fixes

- Fixed bug in *OECalcCartesianCoord* when rotating around the y axis. Prior implementation would give wildly incorrect results.
- Removed dangerous allocation of too small a temporary array for use in writing conformer coordinates to OE-Binary files.
- Fixed bug in `.mol2` file format reader that resulted in connection tables occasionally being skipped when reading files from stdin, pipes and/or sockets. The *OEReadMol2File* function no longer requires the use of tell and seek on its input stream.
- Modified the heuristic in *OERMSD* for recognizing and dealing with degenerate roots. When the heuristic fails, inappropriate rotation matrices were generated.
- Fixed bug in *OESet3DHydrogenGeom* where `memcmp` was comparing coordinates of type `double`, but using `sizeof(float)`, so only a partial comparison was being carried out.
- Modifications which provide the ability to read OEBinary v2 files which contain unrecognized data, without losing the data.
- Added 1 byte at the end of OEMolCT record for molecules with >255 && <65536 bonds to store the endianness of short integers which are used to store bond records. The extra byte won't break old format readers, although oeb files written with old version writers won't be portable. This change doesn't break anything either direction, but does allow for portability of oeb files.

Minor bug fixes

- OEB files now properly store and read conformer energy.
- Fixed bug that prevented OEB's large molecules (>255 atoms) from being cross platform.
- Reading and writing molecules in OEB format no longer fails if a string of 0 length has been tagged to the molecules generic data.
- Fixed bug in *OEMolBase.Sweep* to ensure that atoms/bonds are reindexed sequentially even if there have been no deletions.
- Fixed bug in *OEParseSmiles* that incorrectly parsed atom map indices.
- Fixed bug that *OEAtomBase.SetStereo* was returning `false` even when the stereo was set correctly.
- Fixed a bug in the copy constructor of OEMolTmplt that was only apparent for OEDBMol`s.
- Bug fixes to MDL superatom expansion to avoid problems with the order in which superatoms expanded.
- Fixed the return value of the *DeleteAtom* and *DeleteBond* member functions of *OEMolBase* and `OEMol`. The behavior of these functions was not changed.
- Added missing *IsDeleted(OEAtomBase*)* and *IsDeleted(OEBondBase*)* methods to *OEMolBase*.
- Fixed API misspelling of *OESuppressHydrogens*
- Fixed *OEAddExplicitHydrogens* to return bool rather than void to reflect its failures.
- Fixed *OESet3DHydrogenGeom* to return an appropriate boolean value rather than always returning true.
- In `bool OERMSD (const OEMolBase&, const OEMCMolBase&, double *rmsdArray, const OEMatchBase &match, bool overlay = false, double *rot = 0, double *trans = 0)` fixed bug that the first conformer of the *OEMCMolBase* was used repeatedly rather than using each conformer in succession.
- Corrected behavior of *OESubsetMol* in the case when a ring bond it the only item removed from the original molecule.

- Fixed a bug in *OEChem*'s default *OEAtomBase* implementation that caused problems for an atom after 65536 neighboring bonds had been deleted. This occurred in codes repeatedly created temporary bonds then deleted them without calling *OEAtomBase.Sweep*.
- Fixes to atom naming bugs in *OEChem*'s peptide reside perception routines. Also adds support for the PTR residue, representing phosphotyrosine. Additional fixes to tie splitting of C-terminal serine residues. Use `` H' `` to name C-terminal aldehyde hydrogens (`` H `` denotes the backbone amide nitrogen's hydrogen).
- Changed *OEFuzzy.operator!=* and *OEFuzzy.operator==* to return bool instead of *OEFuzzy*.
- Removed overloaded *OEChem::OEMatch::operator new* as it was incompatible with some versions of STL.

5.93.2 OESystem 1.3.0

New Features

- Added two functions *OESystem::OEFizzGrid::GetFloatsSet* and *OESystem::OEFizzGrid::GetIntegersSet* to determine if the floats and integers are set for this grid.
- Added the new functions *OEBitVector.SetRangeOn* and *OEBitVector.SetRangeOff* to *OEBitVector*.
- Classes to handle grid I/O in OEB files.
- Add default 64 bit file support on Linux.
- Added *OEIter.Sort* member function to iterators. This allows arbitrary sorting of the order in which objects come out of any *OEIter*.
- Added two functions: *OEFileExtension* and *OEFileExtensions* which return a string containing the specified filenames file extension or list of extensions respectively.

Improvements and Optimizations

- *OECheckHelp* now also checks for `--help defaults`, which will list the default arguments of all parameters.
- Added a new function for configuring *OEInterface* which takes an `unsigned char*` rather than a `char*`.
- The *OEWriteSettings* function now defaults to using OEERR rather than OEOUT.
- Added the non-virtual function *OEUnaryPredicate.CreatePredicateCopy* to the *OEUnaryPredicate* and *OEBinaryPredicate* base classes which allow users to use this virtual constructor method without unnecessary casts.

Major bug fixes

- Fixed the previously broken *OEBitVector.FirstBit*, *OEBitVector.LastBit*, *OEBitVector.NextBit*, and *OEBitVector.PrevBit* functions.
- Fixed OEBinary writer bug where atom types were being written as atom names.
- Added handling for unknown data in OEB files for lossless data conversion even when older software versions are unable to parse data.

Minor bug fixes

- Removed a bug where *OESystem::OEFizzGrid::IsDataType* calls *OEBase.IsDataType* as well
- Fixed a bug in OEConcatIter that would cause the iterator to appear to have only part of its contents of two empty iterators were pushed back onto the concat iterator.
- OEB read and write functions now return with meaningful values (previously the writers always returned false regardless and the readers always returned success regardless).

- Fixed problem with some interface parameters calling `new` on zero bytes, then reset the pointer, causing small leaks on some platforms.
- Corrected the error descriptor for FATAL errors.
- Fixed MSVC 7.1 ambiguity in `operator && (const OEIter<T>&, bool)`.
- Added endianness test for short integers to fix problems with small number of OEB files.
- Fixed recurring compile problem in cygwin where `ssize_t` isn't defined.

5.93.3 OEPlatform 1.3.0

New Features

- Added an *OETryMutex* class that allows for an *attempted lock* through the function *OETryMutex.Try*
- Added *OEFileCorrectSeparators* and *OEFileDeterminePathAndName* functions.
- Added new platform-independent *OEIsNaN* function which is overloaded for `float` and `double`. As the name implies, this function returns a boolean which is true if the number passed is a nan.

Improvements and Optimizations

- *OEMutex* implementation now uses pthreads for icc 8.0
- Add default 64 bit file support on Linux.

Major bug fixes

- Defined NONMINMAX in `openeye.h` under win32, which prevents conflicts between STL min/max and windows min/max macros.

Minor bug fixes

- Made gzip set and initialization functions work with std::cin.
- Fixed a bug in the gzip seek function.
- Fixed a few minor issues with the handling of trailing separators in path names.
- Fix bug in *OEPlatform::OEStreamManager::openInput* function. When there was no protocol part of the `filename` parameter, the internal protocol string was not properly NULL-terminated.
- Added code to properly initialize *OEMutex*.
- Used `&&` to replace `and` for more consistent compiler support in *OEMutex*.
- Implemented copy constructor and assignment operator for oeisstream.

CHAPTER

SIX

COPYRIGHT AND TRADEMARKS

© 1997-2024 Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA. All rights reserved.

OpenEye, Lexichem, ROCS, Grapheme, and Orion are registered trademarks of Cadence Design Systems, Inc.

This material contains proprietary information of Cadence Design Systems, Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

The information supplied in this document is believed to be true, but no liability is assumed for its use or the infringement of the rights of others resulting from its use. Information in this document is subject to change without notice and does not represent a commitment on the part of Cadence Design Systems, Inc.

This package is sold/licensed/distributed subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out, or otherwise circulated without Cadence Design Systems' prior consent, in any form of packaging or cover other than that in which it was produced. No part of this manual or accompanying documentation, may be reproduced, stored in a retrieval system on optical or magnetic disk, tape, CD, DVD or other medium, or transmitted in any form or by any means, electronic, mechanical, photocopying recording, or otherwise for any purpose other than for the purchaser's personal use without a legal agreement or other written permission granted by Cadence.

This product should not be used in the planning, construction, maintenance, operation or use of any nuclear facility nor the flight, navigation, or communication of aircraft or ground support equipment. Cadence Design Systems, Inc., shall not be liable, in whole or in part, for any claims arising from such use, including death, bankruptcy or outbreak of war.

Windows is a registered trademark of Microsoft Corporation. Apple, OS X, macOS, and Macintosh are registered trademarks of Apple Inc. UNIX is a registered trademark of the Open Group. Linux is a registered trademark of Linus Torvalds. Red Hat is a registered trademark of Red Hat, Inc. SUSE, SLED, and SLES are registered trademarks of Novell, Inc. Ubuntu is a registered trademark of Canonical Ltd.

SYBYL is a registered trademark of Certara, L.P. MDL, BIOVIA and ISIS are registered trademarks of Dassault Systemes and/or its affiliates. SMARTS and SMIRKS are registered trademarks of Daylight Chemical Information Systems, Inc. MacroModel is a registered trademark of Schrodinger, LLC.

Python is a registered trademark of the Python Software Foundation. Django is a registered trademark of the Django Software Foundation. Java is a registered trademark of Oracle and/or its affiliates.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

CHAPTER

SEVEN

SAMPLE CODE

For all of the code examples and samples within documents published by OpenEye, Cadence Molecular Sciences, the following terms of use apply.

TERMS FOR USE OF SAMPLE CODE

The software below ("Sample Code") is provided to current licensees or subscribers of Cadence products or SaaS offerings (each a "Customer"). Customer is hereby permitted to use, copy, and modify the Sample Code, subject to these terms. Cadence claims no rights to Customer's modifications. Modification of Sample Code is at Customer's sole and exclusive risk. Sample Code may require Customer to have a then current license or subscription to the applicable Cadence offering.

THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. CADENCE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be liable for any damages or liability in connection with the Sample Code or its use.

CHAPTER

EIGHT

CITATION

Please follow the guidelines in this document to cite use of our products in research publications and other documents.

For more information, please contact OpenEye Support.

8.1 Orion®

To cite use of Orion, please use the following:

```
Orion <version-number>. OpenEye, Cadence Molecular Sciences, Santa Fe, NM. http://www.
↪eyesopen.com.
```

For example:

```
Orion 2025.1. OpenEye, Cadence Molecular Sciences, Santa Fe, NM. http://www.eyesopen.com.
```

The version number for Orion appears in the product's release notes, for example, https://docs.eyesopen.com/orion-user-interface/releasenotes/index.html.

8.2 Orion Floes

To cite use of an Orion-based floe package, please use the following:

```
OpenEye <package-name> <version-number>. OpenEye, Cadence Molecular Sciences, Santa Fe,␣
↪NM. http://www.eyesopen.com.
```

For example:

```
OpenEye Large-Scale Floes 4.2.3. OpenEye, Cadence Molecular Sciences, Santa Fe, NM.␣
↪http://www.eyesopen.com.
```

The version number for a floe package is displayed on the first page of the package's release notes. For example: https://docs.eyesopen.com/floe/modules/large_scale_floes/docs/source/releasenotes_2025_1.html.

8.3 Toolkits and Applications

To cite OpenEye toolkits or applications used in your work, please use the following:

```
OpenEye Toolkits [or Applications] <version-number>. OpenEye, Cadence Molecular Sciences,
↪ Santa Fe, NM. http://www.eyesopen.com.
```

For example:

```
OpenEye Toolkits 2024.2.1. OpenEye, Cadence Molecular Sciences, Santa Fe, NM. http://www.
↪eyesopen.com.
```

The Applications and Toolkits are released as a package, multiple times per year. The first part of the version number designates the calendar year. The version number appears on the summary document for the package.

If you want to cite an individual application or toolkit, you can use the syntax:

```
<product name> <version-number>. OpenEye, Cadence Molecular Sciences, Santa Fe, NM.␣
↪http://www.eyesopen.com.
```

where <*product name*> is:

- Bioisostere TK
- OEChem TK
- OEDepict TK
- OEDocking TK
- OEFF TK
- Eon TK
- FastROCS TK
- Grapheme TK
- GraphSim TK
- Lexichem TK
- OEMedChem TK
- MolProp TK
- Omega TK
- Quacpac TK
- Shape TK
- Spicoli TK
- Spruce TK
- Szmap TK
- Szybki TK
- Zap TK
- AFITT Application
- BROOD Application
- OEDocking Application
- EON Application
- OMEGA Application
- PICTO Application
- pKa-Prospector Application
- QUACPAC Application

- ROCS Application
- SiteHopper Application
- SPRUCE Application
- SZMAP Application
- SZYBKI Application
- VIDA Application

You can use the version number of the individual Application or Toolkit. For example:

```
SZYBKI Application 2.8.0.1. OpenEye, Cadence Molecular Sciences, Santa Fe, NM. http://
↪www.eyesopen.com.

Szybki TK 2.8.0.1. OpenEye, Cadence Molecular Sciences, Santa Fe, NM. http://www.
↪eyesopen.com.
```

The first documentation pages give the product names and version numbers, as in https://docs.eyesopen.com/applications/szybki/index.html and https://docs.eyesopen.com/toolkits/python/szybkitk/index.html.

8.4 Publications for Bibliographies

Please also cite these publications if the corresponding product was used in your research.

8.4.1 Orion

Sørensen, J., Bannan, C.C., Calabrò, G., Jain, V., Ovanesyan, G., Smith, A., Zhang, S., Bayly, C.I., Darden, T.A., Geballe, M.T., LeBard, D.N., McGann, M., Moon, J.B., Muddana, H.S., Shewmaker, A., LaFon, J., Tolbert, R.W., Skillman, A.G., Nicholls, A. Orion® A Cloud-Native Molecular Design Platform. In *Computational Drug Discovery*. V. Poongavanam and V. Ramaswamy, Eds. Wiley-VCH, 2024; pp. 579–615. DOI: 10.1002/9783527840748.ch24

8.4.2 AFITT and FLYNN Applications

Wlodek, S.; Skillman, A.G.; Nicholls, A. Automated Ligand Placement and Refinement with a Combined Force Field and Shape Potential. *Acta Crystallogr., Sect. D: Struct. Biol.* **2006**, *62* (7), 741–749. DOI: 10.1107/s0907444906016076

8.4.3 OEDocking Application and Toolkit

Kelley, B.P.; Brown, S.P.; Warren, G.L.; Muchmore, S.W. POSIT: Flexible Shape-Guided Docking For Pose Prediction. *J. Chem. Inf. Model.* **2015**, *55* (8), 1771-1780. DOI: 10.1021/acs.jcim.5b00142

McGann, M. FRED Pose Prediction and Virtual Screening Accuracy. *J. Chem. Inf. Model.* **2011**, *51* (3), 578-596. DOI: 10.1021/ci100436p

McGann, M. FRED and HYBRID Docking Performance on Standardized Datasets. *J. Comput.-Aided Mol. Des.* **2012**, *26* (8), 897-906. DOI: 10.1007/s10822-012-9584-8

8.4.4 OMEGA Application and Toolkit

If you are using OMEGA in torsion driving mode, please cite the following two publications.

Hawkins, P.C.D.; Skillman, A.G.; Warren, G.L.; Ellingson, B.A.; Stahl, M.T. Conformer Generation with OMEGA: Algorithm and Validation Using High Quality Structures from the Protein Databank and the Cambridge Structural Database. *J. Chem. Inf. Model.* **2010**, *50* (4), 572-584. DOI: 10.1021/ci100031x

Hawkins, P.C.D. and Nicholls, A. Conformer Generation with OMEGA: Learning from the Data Set and the Analysis of Failures. *J. Chem. Inf. Model.* **2012**, *52* (11), 2919–2936. DOI: /10.1021/ci300314k

If you are using OMEGA in macrocyle mode, please cite the following.

Hawkins, P.C.D. and Wlodek, S. Decisions with Confidence: Application to the Conformation Sampling of Molecules in the Solid State. *J. Chem. Inf. Model.* **2020**, *60* (7), 3518–3533. DOI: /10.1021/acs.jcim.0c00358

8.4.5 ROCS Application

Hawkins, P.C.D.; Skillman, A.G.; Nicholls, A. Comparison of Shape-Matching and Docking as Virtual Screening Tools. *J. Med. Chem.* **2007**, *50* (1), 74-82. DOI: 10.1021/jm0603365

8.5 OpenEye MMDS Web Service

To cite use of the Macromolecular Data Service (MMDS) web service, please use the syntax:

```
Macromolecular Data Service <version-number>. OpenEye, Cadence Molecular Sciences, Santa␣
↪Fe, NM. http://www.eyesopen.com.
```

For example:

```
Macromolecular Data Service 1.1. OpenEye, Cadence Molecular Sciences, Santa Fe, NM.␣
↪http://www.eyesopen.com.
```

The MMDS version number appears on the web service's release notes.

CHAPTER

NINE

TECHNOLOGY LICENSING

OpenEye products use the following technology under license.

Some of the open source technologies we use are licensed under the GNU Public License (GPL) version 2 or 3. In all instances, these technologies are communicated with using the "at arms length" principle, using pipes and command line arguments.

While the software is in some instances assembled into a container, that assembly does not invoke the GPL copyleft scope. For each Floe package where these are used, the environment log clearly details the installed programs and their version numbers. With the links below for each technology, the source for each of those programs is available.

Table 1: Licensed Technology

Name of Project	Website	License URL
_libgcc_mutex	https://github.com/conda-forge/_libgcc_mutex-feedstock	https://github.com/conda-forge/_libgcc_mutex-feedstock/blob/master/LICENSE.txt
_openmp_mutex	https://github.com/conda-forge/_openmp_mutex-feedstock	https://github.com/conda-forge/_openmp_mutex-feedstock/blob/main/LICENSE.txt
ablang2	https://github.com/TobiasHeOl/AbLang2	https://github.com/TobiasHeOl/AbLang2/blob/main/LICENSE
abseil-cpp	https://github.com/abseil/abseil-cpp	https://github.com/abseil/abseil-cpp/blob/master/LICENSE
absl-py	https://github.com/abseil/abseil-py	https://github.com/abseil/abseil-py/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
aiohappyeyeballs	https://github.com/aio-libs/aiohappyeyeballs	https://github.com/aio-libs/aiohappyeyeballs/blob/main/LICENSE
aiohttp	https://docs.aiohttp.org/en/stable/	https://github.com/aio-libs/aiohttp/blob/master/LICENSE.txt
aiosignal	https://github.com/aio-libs/aiosignal	https://github.com/aio-libs/aiosignal/blob/master/LICENSE
Amazon Linux 2	https://aws.amazon.com/amazon-linux-2	N/A
AmberUtils	http://ambermd.org	N/A
ambit	https://github.com/khimaros/ambit	https://github.com/khimaros/ambit/blob/master/LICENSE
amqp	https://github.com/celery/py-amqp	https://github.com/celery/py-amqp/blob/master/LICENSE
anaconda-anon-usage	https://github.com/anaconda/anaconda-anon-usage	https://github.com/anaconda/anaconda-anon-usage/blob/main/LICENSE
angular	https://github.com/angular/angular.js	https://github.com/angular/angular.js/raw/master/LICENSE.md
angular-animate	https://github.com/angular/angular.js	https://github.com/angular/angular.js/raw/master/LICENSE.md
angular-cache	https://github.com/jmdobry/angular-cache	https://github.com/jmdobry/angular-cache/raw/master/LICENSE
angular-cookies	https://github.com/angular/angular.js	https://github.com/angular/angular.js/raw/master/LICENSE.md

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
angular-loggly-logger	https://github.com/ajbrown/angular-loggly-logger	https://github.com/ajbrown/angular-loggly-logger/raw/master/LICENSE.md
angular-mocks	https://github.com/angular/angular.js	https://raw.githubusercontent.com/angular/angular.js/master/LICENSE
angular-resource	https://github.com/angular/angular.js	https://raw.githubusercontent.com/angular/angular.js/master/LICENSE
angular-toggle-switch	https://github.com/cgarvis/angular-toggle-switch	https://github.com/cgarvis/angular-toggle-switch/blob/master/LICENSE
angular-ui-grid	https://github.com/angular-ui/ui-grid	https://github.com/angular-ui/ui-grid
angular-ui-router	https://github.com/angular-ui/ui-router	https://github.com/angular-ui/ui-router/raw/master/LICENSE
angular-ui-tree	https://github.com/angular-ui-tree/angular-ui-tree	https://github.com/angular-ui-tree/angular-ui-tree/raw/master/LICENSE
angular-vs-repeat	https://github.com/kamilkp/angular-vs-repeat	https://raw.githubusercontent.com/kamilkp/angular-vs-repeat/master/LICENSE
aniso8601	https://bitbucket.org/nielsenb/aniso8601	https://bitbucket.org/nielsenb/aniso8601/src/master/LICENSE
annotated-types	https://github.com/annotated-types/annotated-types	https://github.com/annotated-types/annotated-types/blob/main/LICENSE
antlr4-python3-runtime	http://www.antlr.org OR https://pypi.org/project/antlr4-python3-runtin	https://github.com/conda-forge/antlr4-python3-runtin/blob/master/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
anyio	https://github.com/agronholm/anyio	https://github.com/agronholm/anyio/blob/master/LICENSE
appdirs	http://github.com/ActiveState/appdirs	http://github.com/ActiveState/appdirs/blob/master/LICENSE.txt
appengine	https://google.golang.org/appengine	https://github.com/golang/appengine/blob/master/LICENSE
arabic-reshaper	https://github.com/mpcabd/python-arabic-reshape	https://github.com/mpcabd/python-arabic-reshape/blob/master/LICENSE
archspec	https://github.com/archspec/archspec/blob/master/README.md	https://github.com/archspec/archspec/blob/master/COPYRIGHT
argon2-cffi	https://github.com/hynek/argon2-cffi	https://github.com/hynek/argon2-cffi/blob/master/LICENSE
argon2-cffi-bindings	https://github.com/hynek/argon2-cffi-bindings	https://github.com/hynek/argon2-cffi-bindings/blob/master/LICENSE
arpack	https://www.caam.rice.edu/software/ARPACK/	https://www.caam.rice.edu/software/ARPACK/
arrow	https://github.com/arrow-py/arrow	https://github.com/arrow-py/arrow?tab=Apache-2.0-1-ov-file#readme
ascii85	https://github.com/huandu/node-ascii85	https://github.com/huandu/node-ascii85/raw/master/LICENSE
ase	https://wiki.fysik.dtu.dk/ase/	https://wiki.fysik.dtu.dk/ase/development/licenseinfo.html#license-info
asgiref	https://github.com/django/asgiref/	https://github.com/django/asgiref/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
asn1crypto	https://github.com/wbond/asn1crypto	https://github.com/wbond/asn1crypto/blob/master/LICENSE
assertions go-render	https://github.com/smartystreets/assertions/internal/go-render/render	https://github.com/smartystreets/assertions/blob/master/internal/go-render/LICENSE
assertions oglmatchers	https://github.com/smartystreets/assertions/internal/oglematchers	https://github.com/smartystreets/assertions/blob/master/internal/oglematchers/LICENSE
assertions	https://github.com/smartystreets/assertions	https://github.com/smartystreets/assertions/blob/master/LICENSE.md
asttokens	https://github.com/gristlabs/asttokens	https://github.com/gristlabs/asttokens/blob/master/LICENSE
astunparse	https://github.com/simonpercivall/astunparse	https://github.com/simonpercivall/astunparse/blob/master/LICENSE
async-lru	https://github.com/aio-libs/async-lru	https://github.com/aio-libs/async-lru/blob/master/LICENSE
async-timeout	https://github.com/aio-libs/async-timeout	https://github.com/aio-libs/async-timeout/blob/master/LICENSE
atk-1.0	https://docs.gtk.org/atk/	https://gitlab.gnome.org/GNOME/atk/-/blob/master/COPYING
atomic	https://github.com/uber-go/atomic	https://github.com/uber-go/atomic/blob/master/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
atomicwrites	https://github.com/untitaker/python-atomicwrites	https://github.com/untitaker/python-atomicwrites/blob/master/LICENSE
attrs	https://www.attrs.org/	https://github.com/python-attrs/attrs/blob/main/LICENSE
aws-sdk-go	https://github.com/aws/aws-sdk-go	https://github.com/aws/aws-sdk-go/blob/master/LICENSE.txt
babel	https://babel.pocoo.org/ OR https://github.com/python-babel/babel	https://github.com/python-babel/babel/blob/master/LICENSE OR https://babel.pocoo.org/en/latest/license.html#babel-license
Babel	https://github.com/python-babel/babel	https://github.com/python-babel/babel/blob/master/LICENSE
backcall	https://github.com/takluyver/backcall	https://github.com/takluyver/backcall/blob/master/LICENSE
backports	https://github.com/brandon-rhodes/backports	https://github.com/brandon-rhodes/backports/blob/master/LICENSE.TXT
backports.functools-lru-cache	https://github.com/jaraco/backports.functools_lru_cache	https://github.com/jaraco/backports.functools_lru_cache/blob/main/LICENSE
backports.zstd	https://github.com/rogdham/backports.zstd	https://github.com/rogdham/backports.zstd?tab=License-1-ov-file#readme
base62	https://github.com/kare/base62	https://github.com/kare/base62/blob/main/LICENSE
beautifulsoup4	https://www.crummy.com/software/BeautifulSoup/	https://www.crummy.com/software/BeautifulSoup/

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
billiard	https://github.com/celery/billiard	https://github.com/celery/billiard/blob/master/LICENSE.txt
biopython	https://biopython.org	https://github.com/biopython/biopython/blob/master/LICENSE.rst
biotite	https://github.com/biotite-dev/biotite/blob/master/README.rst	https://github.com/biotite-dev/biotite/blob/master/LICENSE.rst
bitarray	https://github.com/ilanschnell/bitarray	https://github.com/ilanschnell/bitarray/blob/master/LICENSE
bitset	https://github.com/willf/bitset	https://github.com/willf/bitset/blob/master/LICENSE
blas	https://www.netlib.org/blas/	https://www.netlib.org/blas/#_licensing
bleach	https://github.com/mozilla/bleach	https://github.com/mozilla/bleach/blob/main/LICENSE
blessings	https://github.com/erikrose/blessings	https://github.com/erikrose/blessings/blob/master/LICENSE
blinker	https://pythonhosted.org/blinker/	https://github.com/pallets-eco/blinker/blob/master/LICENSE
bokeh	https://github.com/bokeh/bokeh	https://github.com/bokeh/bokeh/blob/branch-4.0/LICENSE.txt
blosc	https://github.com/Blosc/python-blosc	https://github.com/Blosc/python-blosc/blob/master/LICENSE.txt
blosc2	https://github.com/Blosc/python-blosc2	https://github.com/Blosc/c-blosc2/blob/main/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
boltons	https://github.com/mahmoud/boltons	https://github.com/mahmoud/boltons/blob/master/LICENSE
Boltz	https://boltz.bio	https://github.com/jwohlwend/boltz/blob/main/LICENSE
boost	https://www.boost.org/doc/libs/1_71_0/libs/python/doc/html/index.html	https://www.boost.org/LICENSE_1_0.txt
boost-cpp	https://www.boost.org/doc/libs/1_71_0/libs/python/doc/html/index.html	https://www.boost.org/LICENSE_1_0.txt
bootstrap-vue	https://github.com/bootstrap-vue/bootstrap-vue	https://github.com/bootstrap-vue/bootstrap-vue/raw/master/LICENSE
boto3	https://github.com/boto/boto3	https://github.com/boto/boto3/blob/develop/LICENSE
botocore	https://github.com/boto/botocore	https://github.com/boto/botocore/blob/develop/LICENSE.txt
Bottleneck	https://bottleneck.readthedocs.io/en/latest/index.html	https://bottleneck.readthedocs.io/en/latest/license.html
Brotli	https://github.com/google/brotli	https://github.com/google/brotli/blob/master/LICENSE
brotli-bin	https://github.com/google/brotli	https://github.com/google/brotli/blob/master/LICENSE
brotli-python	http://python-hyper.org/projects/brotlipy/en/latest/	https://github.com/python-hyper/brotlicffi/blob/main/LICENSE
brotlicffi	https://github.com/python-hyper/brotlicffi	https://github.com/python-hyper/brotlicffi?tab=MIT-1-ov-file#readme
brotlipy	https://github.com/python-hyper/brotlicffi	https://github.com/python-hyper/brotlicffi/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
bson	http://github.com/py-bson/bson	https://github.com/py-bson/bson/blob/master/LICENSE_APACHE
bytefmt	https://code.cloudfoundry.org/bytefmt	https://github.com/cloudfoundry/bytefmt/blob/master/LICENSE
bzip2	https://www.bzip.org	https://spdx.org/licenses/bzip2-1.0.6.html
c-ares	https://github.com/c-ares/c-ares	https://github.com/c-ares/c-ares/blob/main/LICENSE.md
ca-certificates	https://github.com/conda-forge/ca-certificates-feedsto	https://wiki.mozilla.org/CA/Included_Certificates
cached-property	https://github.com/pydanny/cached-property	https://github.com/pydanny/cached-property/blob/master/LICENSE
cachetools	https://github.com/tkem/cachetools/	https://github.com/tkem/cachetools/blob/master/LICENSE
cairo	https://pycairo.readthedocs.io/en/latest/	https://github.com/pygobject/pycairo/blob/master/COPYING
canvas	https://github.com/Automattic/node-canvas	https://github.com/Automattic/node-canvas
cctbx	https://github.com/cctbx/cctbx_project	https://github.com/cctbx/cctbx_project/blob/master/cctbx/LICENSE_2_0.txt
celery	https://github.com/celery/celery	https://github.com/celery/celery/blob/master/LICENSE
Cerberus	https://docs.python-cerberus.org/en/stable/	https://docs.python-cerberus.org/en/stable/license.html
certifi	https://certifiio.readthedocs.io/en/latest/	https://wiki.mozilla.org/index.php?title=CA/IncludedCertificates&redirect=no

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
cffi	https://github.com/python-cffi	https://github.com/cffi/cffi/blob/master/COPYRIGHT
cftime	https://pypi.org/project/cftime/	https://github.com/Unidata/cftime/blob/master/LICENSE
chardet	https://github.com/chardet/chardet	https://github.com/chardet/chardet/blob/master/LICENSE
charset-normalizer	https://github.com/ousret/charset_normalizer	https://github.com/Ousret/charset_normalizer/blob/master/LICENSE
ChEMBL Structure Pipeline	https://github.com/chembl/ChEMBL_Structure_Pipeline/tree/master	https://github.com/chembl/ChEMBL_Structure_Pipeline/blob/master/LICENSE
choreographer	https://github.com/mazpie/choreographer	https://github.com/mazpie/choreographer/blob/main/LICENSE
chunkreader	https://github.com/jackc/chunkreader/v2	https://github.com/jackc/chunkreader/blob/master/LICENSE
click	https://palletsprojects.com/p/click/	https://github.com/pallets/click/blob/main/LICENSE.rst
click-completion	https://github.com/click-contrib/click-completion	https://github.com/click-contrib/click-completion/blob/master/LICENSE
click-didyoumean	https://github.com/click-contrib/click-didyoumean	https://github.com/click-contrib/click-didyoumean/blob/master/LICENSE
click-plugins	https://github.com/click-contrib/click-plugins	https://github.com/click-contrib/click-plugins/blob/master/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
click-repl	https://github.com/untitaker/click-repl	https://github.com/click-contrib/click-repl/blob/master/LICENSE
cloudpickle	https://github.com/cloudpipe/cloudpickle	https://github.com/cloudpipe/cloudpickle/blob/master/LICENSE
cmake	https://cmake.org/	https://cmake.org/licensing/
colorama	https://github.com/tartley/colorama	https://github.com/tartley/colorama/blob/master/LICENSE.txt
comm	https://github.com/ipython/comm	https://github.com/ipython/comm/blob/master/LICENSE
compose	https://github.com/docker/compose	https://github.com/docker/compose/blob/v2/LICENSE
conda-content-trust	https://github.com/conda/conda-content-trust	https://github.com/conda/conda-content-trust/blob/main/LICENSE
conda-libmamba-solver	https://github.com/conda/conda-libmamba-solv	https://github.com/conda/conda-libmamba-solv blob/main/LICENSE
conda-pack	https://conda.github.io/conda-pack/	https://github.com/conda/conda-pack/blob/main/LICENSE.txt
conda-package-handling	https://github.com/conda/conda-package-handli	https://github.com/conda/conda-package-handli blob/main/LICENSE
conda-package-streaming	https://anaconda.org/conda-forge/conda-package-stream	https://github.com/conda/conda-package-stream blob/main/LICENSE
conda-token	https://anaconda.org/anaconda/conda-token	https://anaconda.org/anaconda/conda-token

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
conda_package_strear	https://github.com/conda/conda-package-streamtree/main/conda_package_streaming	https://github.com/conda/conda-package-streamtab=License-1-ov-file
confuse	https://github.com/beetbox/confuse	https://github.com/beetbox/confuse/blob/main/LICENSE
contourpy	https://github.com/contourpy/contourpy	https://github.com/contourpy/contourpy/blob/main/LICENSE
cpp-expected	https://github.com/TartanLlama/expected	https://github.com/TartanLlama/expected?tab=CC0-1.0-1-ov-file#readme
cpp-peglib	https://github.com/yhirose/cpp-peglib	https://github.com/yhirose/cpp-peglib/blob/master/LICENSE
cryptography	https://github.com/pyca/cryptography	https://github.com/pyca/cryptography/blob/main/LICENSE
cssselect2	https://github.com/Kozea/cssselect2/	https://github.com/Kozea/cssselect2/blob/master/LICENSE
cuda-bindings	https://nvidia.github.io/cuda-python/cuda-bindings/latest/install.html	https://github.com/NVIDIA/cuda-python/blob/main/cuda_bindings/LICENSE
cuda-pathfinder	https://nvidia.github.io/cuda-python/cuda-pathfinder/latest/	https://nvidia.github.io/cuda-python/cuda-pathfinder/latest/license.html
cudatoolkit	https://developer.nvidia.com/cuda-toolkit	https://docs.nvidia.com/cuda/eula/index.html
cuequivariance-ops-cu12	https://developer.nvidia.com/cuequivariance	https://github.com/NVIDIA/cuEquivariance/blob/main/LICENSES/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
cuequivariance-ops-torch-cu12	https://developer.nvidia.com/cuequivariance	https://github.com/NVIDIA/cuEquivariance/blob/main/LICENSES/LICENSE
cuequivariance-torch	https://docs.nvidia.com/cuda/cuequivariance/	https://github.com/NVIDIA/cuEquivariance/blob/main/LICENSES/LICENSE
cupy-cuda113	https://cupy.dev/	https://github.com/cupy/cupy/blob/main/LICENSE
cupy-cuda11x	https://cupy.dev/	https://github.com/cupy/cupy/?tab=MIT-1-ov-file#readme
curl	https://curl.se	https://curl.se/docs/copyright.html
cycler	https://github.com/matplotlib/cycler	https://github.com/matplotlib/cycler/blob/main/LICENSE
cyrus-sasl	https://github.com/cyrusimap/cyrus-sasl	https://github.com/cyrusimap/cyrus-sasl/blob/master/COPYING
Cython	https://cython.org	https://www.apache.org/licenses/LICENSE-2.0.html
d3	https://github.com/mbostock/d3	https://github.com/mbostock/d3/raw/master/LICENSE
dataclasses	https://github.com/ericvsmith/dataclasses	https://github.com/ericvsmith/dataclasses/blob/master/LICENSE.txt
ddsketch	http://github.com/datadog/sketches-py	https://github.com/DataDog/sketches-py/blob/master/LICENSE
debugpy	https://aka.ms/debugpy	https://github.com/microsoft/debugpy/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
decimal	https://github.com/shopspring/decimal	https://github.com/shopspring/decimal/blob/master/LICENSE
decorator	https://github.com/micheles/decorator	https://github.com/micheles/decorator/blob/master/LICENSE.txt
deepdiff	https://github.com/seperman/deepdiff	https://github.com/seperman/deepdiff/blob/master/LICENSE
deeptime	https://github.com/deeptime-ml/deeptime	https://github.com/deeptime-ml/deeptime/blob/main/LICENSE.txt
defusedxml	https://github.com/tiran/defusedxml	https://github.com/tiran/defusedxml/blob/main/LICENSE
dftd3	https://github.com/dftd3/tad-dftd3	https://github.com/dftd3/tad-dftd3/blob/main/LICENSE
dill	https://github.com/uqfoundation/dill	https://github.com/uqfoundation/dill/blob/master/LICENSE
distlib	https://distlib.readthedocs.io/en/latest/	https://github.com/pypa/distlib/blob/master/LICENSE.txt
distro	https://github.com/python-distro/distro	https://github.com/python-distro/distro/blob/master/LICENSE
Django	https://www.djangoproject.com/	https://github.com/django/django/blob/main/LICENSE
django-classy-tags	https://github.com/django-cms/django-classy-tags	https://github.com/django-cms/django-classy-tags/blob/master/LICENSE
django-cors-headers	https://github.com/adamchainz/django-cors-headers	https://github.com/adamchainz/django-cors-headers/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
django-csp	https://github.com/mozilla/django-csp	https://github.com/mozilla/django-csp/blob/main/LICENSE
django-extensions	https://github.com/django-extensions/django-extensions	https://github.com/django-extensions/django-extensions/blob/main/LICENSE
django-filter	https://github.com/carltongibson/django-filter/tree/master	https://github.com/carltongibson/django-filter/blob/main/LICENSE
django-redis	https://github.com/jazzband/django-redis	https://github.com/jazzband/django-redis/blob/master/LICENSE
django-taggit	https://github.com/jazzband/django-taggit	https://github.com/jazzband/django-taggit/blob/master/LICENSE
django-taggit-serializer	https://github.com/glemmaPaul/django-taggit-serializ	https://github.com/glemmaPaul/django-taggit-serializ blob/master/LICENSE
django-taggit-templatetags2	https://github.com/fizista/django-taggit-templat	https://github.com/fizista/django-taggit-templat blob/master/LICENSE.txt
djangorestframework	https://www.django-rest-framewor org/	https://github.com/encode/django-rest-framewor blob/master/LICENSE.md
dkh	https://psicode.org/psi4manual/master/dkh.html	https://github.com/psi4/dkh/blob/master/COPYING.LESSER
dm-tree	https://github.com/deepmind/tree	https://github.com/deepmind/tree/blob/master/LICENSE
docker-py	https://github.com/docker/docker-py/	https://github.com/docker/docker-py/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
docker-pycreds	https://github.com/shin-/dockerpy-creds	https://github.com/shin-/dockerpy-creds/blob/master/LICENSE
docopt	https://docopt.org	https://github.com/docopt/docopt/blob/master/LICENSE-MIT
docutils	https://docutils.sourceforge.io	https://docutils.sourceforge.io/COPYING.html
drf-dynamic-fields	https://github.com/dbrgn/drf-dynamic-fields	https://github.com/dbrgn/drf-dynamic-fields/blob/master/LICENSE
DuckDB	https://duckdb.org/	https://github.com/duckdb/duckdb?tab=MIT-1-ov-file#readme
edgembar	https://rutgerslbsr.gitlab.io/fe-toolkit/edgembar/index.html	https://gitlab.com/RutgersLBSR/fe-toolkit/-/blob/master/LICENSE
editdistance	https://github.com/roy-ht/editdistance	https://github.com/roy-ht/editdistance/blob/master/LICENSE
eigen	https://eigen.tuxfamily.org/index.php?title=Main_Page	https://eigen.tuxfamily.org/index.php?title=Main_Page#License
einops	https://github.com/arogozhnikov/einops	https://github.com/arogozhnikov/einops/blob/master/LICENSE
einx	https://github.com/fferflo/einx	https://github.com/fferflo/einx/blob/master/LICENSE
entrypoints	https://github.com/takluyver/entrypoints	https://github.com/takluyver/entrypoints/blob/master/LICENSE
errors	https://github.com/pkg/errors	https://github.com/pkg/errors/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
eslint-plugin	https://github.com/typescript-eslint/typescript-eslint	https://github.com/typescript-eslint/typescript-eslint/raw/master/LICENSE
et_xmlfile	https://foss.heptapod.net/openpyxl/et_xmlfile	https://foss.heptapod.net/openpyxl/et_xmlfile/-/blob/branch/default/LICENCE.rst
exceptiongroup	https://github.com/agronholm/exceptiongroup	https://github.com/agronholm/exceptiongroup/blob/main/LICENSE
execnet	https://execnet.readthedocs.io/en/latest/	https://github.com/pytest-dev/execnet/blob/master/LICENSE
executing	https://github.com/alexmojaki/executing	https://github.com/alexmojaki/executing/blob/master/LICENSE.txt
expat	https://libexpat.github.io	https://github.com/libexpat/libexpat/blob/master/expat/COPYING
fairscale	https://github.com/facebookresearch/fairscale	https://github.com/facebookresearch/fairscale/blob/main/LICENSE
fairseq	https://github.com/pytorch/fairseq	https://github.com/facebookresearch/fairseq/blob/main/LICENSE
fastjsonschema	https://github.com/horejsek/python-fastjsonschem	https://github.com/horejsek/python-fastjsonschem/blob/master/LICENSE
fastrlock	https://github.com/scoder/fastrlock	https://github.com/scoder/fastrlock/blob/master/LICENSE
fftw	https://www.fftw.org	commercial license

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
filebuffer	https://github.com/mattetti/filebuffer	https://github.com/mattetti/filebuffer/blob/master/LICENSE
filelock	https://py-filelock.readthedocs.io/en/latest/index.html	https://github.com/tox-dev/py-filelock/blob/main/LICENSE
finufft	https://github.com/flatironinstitute/finufft	https://github.com/flatironinstitute/finufft/blob/master/LICENSE
Flask	https://flask.palletsprojects.com/en/2.1.x/	https://flask.palletsprojects.com/en/2.1.x/license/
flatbuffers	https://google.github.io/flatbuffers/	https://github.com/google/flatbuffers/blob/master/LICENSE.txt
flexcache	https://github.com/hgrecco/flexcache	https://github.com/hgrecco/flexcache/blob/main/LICENSE
flexparser	https://github.com/hgrecco/flexparser	https://github.com/hgrecco/flexparser/blob/main/LICENSE
flit-core	https://github.com/pypa/flit	https://github.com/scientific-python/lazy_loader/blob/main/LICENSE.md
FLTK	https://www.fltk.org/index.php	https://www.fltk.org/COPYING.php
fmt	https://fmt.dev/latest/index.html	https://github.com/fmtlib/fmt/blob/master/LICENSE.rst
font-awesome	https://github.com/FortAwesome/Font-Awesome	https://github.com/FortAwesome/Font-Awesome
font-ttf-dejavu-sans-mono	https://dejavu-fonts.github.io	https://dejavu-fonts.github.io/License.html
font-ttf-inconsolata	https://fonts.google.com/specimen/Inconsolata	https://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=OFL

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
font-ttf-source-code-pro	https://fonts.google.com/specimen/Source+Code+Pro	https://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=OFL
font-ttf-ubuntu	https://fonts.google.com/specimen/Ubuntu	https://ubuntu.com/legal/font-licence
fontconfig	https://www.freedesktop.org/wiki/Software/fontconfig/	https://gitlab.freedesktop.org/fontconfig/fontconfig/-/blob/main/COPYING
fonts-conda-ecosystem	https://anaconda.org/conda-forge/fonts-conda-ecosyster	https://docs.conda.io/en/latest/license.html
fonts-conda-forge	https://anaconda.org/conda-forge/fonts-conda-forge/	https://creativecommons.org/licenses/by/4.0/
fonttools	https://github.com/fonttools/fonttools	https://github.com/fonttools/fonttools/blob/main/LICENSE
forcebalance	https://simtk.org/home/forcebalance OR https://github.com/leeping/forcebalance	https://github.com/leeping/forcebalance/blob/master/LICENSE
fqdn	https://github.com/ypcrts/fqdn	https://github.com/ypcrts/fqdn?tab=MPL-2.0-1-ov-file#readme
freetype	https://freetype.org	https://freetype.org/license.html
freetype-py	https://github.com/rougier/freetype-py	https://github.com/rougier/freetype-py?tab=License-1-ov-file#readme
fribidi	https://github.com/fribidi/fribidi	https://github.com/fribidi/fribidi/blob/master/COPYING
frozendict	https://github.com/Marco-Sulla/python-frozendict	https://github.com/Marco-Sulla/python-frozendict/blob/master/LICENSE.txt
frozenlist	https://github.com/aio-libs/frozenlist	https://github.com/aio-libs/frozenlist/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
fsmlite	https://github.com/tkem/fsmlite	https://github.com/tkem/fsmlite/blob/master/LICENSE
fsspec	https://github.com/fsspec/filesystem_spec	https://github.com/fsspec/filesystem_spec/blob/master/LICENSE
funcy	https://github.com/Suor/funcy	https://github.com/Suor/funcy/blob/master/LICENSE
gast	https://github.com/serge-sans-paille/gast/	https://github.com/serge-sans-paille/gast/blob/master/LICENSE
gau2grid	https://github.com/dgasmith/gau2grid	https://github.com/dgasmith/gau2grid/blob/master/LICENSE
gax-go	https://github.com/googleapis/gax-go/v2	https://github.com/googleapis/gax-go/blob/main/LICENSE
gdk-pixbuf	https://gitlab.gnome.org/GNOME/gdk-pixbuf	https://gitlab.gnome.org/GNOME/gdk-pixbuf/-/blob/master/COPYING
gemmi	https://gemmi.readthedocs.io/en/latest/	https://github.com/project-gemmi/gemmi/blob/master/LICENSE.txt
genproto	https://google.golang.org/genproto/googleapis	https://github.com/googleapis/go-genproto/blob/main/LICENSE
geometric	https://openbase.com/python/geometric	https://openbase.com/python/geometric
giflib	https://giflib.sourceforge.net	https://github.com/mirrorer/giflib/blob/master/COPYING
gitdb	https://github.com/gitpython-developers/gitdb	https://github.com/gitpython-developers/gitdb/blob/master/LICENSE
GitPython	https://github.com/gitpython-developers/GitPython	https://github.com/gitpython-developers/GitPython/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
glib	https://docs.gtk.org/glib/	https://gitlab.gnome.org/GNOME/glib/-/blob/main/LICENSES/LGPL-2.1-or-later.txt
GLM Library	https://github.com/g-truc/glm	https://github.com/g-truc/glm/blob/master/manual.md#section0
gls	https://github.com/jtolds/gls	https://github.com/jtolds/gls/blob/master/LICENSE
gmpy2	https://github.com/aleaxit/gmpy	https://github.com/gmpy2/gmpy2/blob/master/COPYING.LESSER
go-cleanhttp	https://github.com/hashicorp/go-cleanhttp	https://github.com/hashicorp/go-cleanhttp/blob/master/LICENSE
go-connections	https://github.com/docker/go-connections	https://github.com/docker/go-connections/blob/master/LICENSE
go-cpio	https://github.com/cavaliercoder/go-cpio	https://github.com/cavaliercoder/go-cpio/blob/master/LICENSE
go-getter	https://github.com/hashicorp/go-getter	https://github.com/hashicorp/go-getter/blob/master/LICENSE
go-homedir	https://github.com/mitchellh/go-homedir	https://github.com/mitchellh/go-homedir/blob/master/LICENSE
go-ini	https://github.com/go-ini/ini	https://github.com/go-ini/ini/blob/master/LICENSE
go-jmespath	https://github.com/jmespath/go-jmespath	https://github.com/jmespath/go-jmespath/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
go-junit-report	https://github.com/jstemmer/go-junit-report	https://github.com/jstemmer/go-junit-report/blob/master/LICENSE
go-netrc	https://github.com/bgentry/go-netrc/netrc	https://github.com/bgentry/go-netrc/blob/master/LICENSE
go-ole	https://github.com/go-ole/go-ole	https://github.com/go-ole/go-ole/blob/master/LICENSE
go-pg	https://github.com/go-pg/pg	https://github.com/go-pg/pg/blob/master/LICENSE
go-redis	https://github.com/go-redis/redis/v8	https://github.com/go-redis/redis/blob/master/LICENSE
go-rendezvous	https://github.com/dgryski/go-rendezvous	https://github.com/dgryski/go-rendezvous/blob/master/LICENSE
go-safetemp	https://github.com/hashicorp/go-safetemp	https://github.com/hashicorp/go-safetemp/blob/master/LICENSE
go-sysconf	https://github.com/tklauser/go-sysconf	https://github.com/tklauser/go-sysconf/blob/master/LICENSE
go-testing-interface	https://github.com/mitchellh/go-testing-interface	https://github.com/mitchellh/go-testing-interface/blob/master/LICENSE
go-units	https://github.com/docker/go-units	https://github.com/docker/go-units/blob/master/LICENSE
go-version	https://github.com/hashicorp/go-version	https://github.com/hashicorp/go-version/blob/master/LICENSE
go-zglob	https://github.com/mattn/go-zglob	https://github.com/mattn/go-zglob/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
go.opencensus	https://go.opencensus.io	https://github.com/census-instrumentatio opencensus-go/blob/master/LICENSE
gobrake.v2	https://gopkg.in/airbrake/gobrake.v2	https://github.com/airbrake/gobrake/blob/v2.0.9/LICENSE
goconvey	https://github.com/smartystreets/goconvey	https://github.com/smartystreets/goconvey/blob/master/LICENSE.md
golden-layout	https://github.com/deepstreamIO/golden-layout	https://github.com/deepstreamIO/golden-layout/raw/master/LICENSE
google-auth	https://google-auth.readthedocs.io/en/master/	https://github.com/GoogleCloudPlatform google-auth-library-p blob/master/LICENSE
google-auth-oauthlib	https://github.com/googleapis/google-auth-library-p	https://github.com/googleapis/google-auth-library-p blob/main/LICENSE
google-cloud-go	https://cloud.google.com/go	https://github.com/googleapis/google-cloud-go/blob/main/LICENSE
google-cloud-go/storage	https://cloud.google.com/go/storage	https://github.com/googleapis/google-cloud-go/blob/main/LICENSE
google-pasta	https://github.com/google/pasta	https://github.com/google/pasta/blob/master/LICENSE
google.golang.org/api	https://google.golang.org/api	https://github.com/googleapis/google-api-go-client/blob/main/LICENSE
gopsutil	https://github.com/shirou/gopsutil	https://github.com/shirou/gopsutil/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
gorilla websocket	https://github.com/gorilla/websocket	https://github.com/gorilla/websocket/blob/master/LICENSE
gpytorch	https://gpytorch.ai	https://github.com/cornellius-gp/gpytorch?tab=MIT-1-ov-file#readme
graphite2	https://github.com/silnrsi/graphite	https://github.com/silnrsi/graphite/blob/master/LICENSE
graphviz	https://graphviz.org/	https://graphviz.org/license/
greenlet	https://greenlet.readthedocs.io/en/latest/	https://github.com/python-greenlet/greenlet/blob/master/LICENSE
groupcache	https://github.com/golang/groupcache	https://github.com/golang/groupcache/blob/master/LICENSE
grpc	https://google.golang.org/grpc	https://github.com/grpc/grpc-go/blob/master/LICENSE
grpc-cpp	https://github.com/grpc/grpc	https://github.com/grpc/grpc/blob/master/LICENSE
grpcio	https://github.com/grpc/grpc.io/blob/main/LICENSE	https://github.com/grpc/grpc.io/blob/main/LICENSE
gtk2	https://gitlab.gnome.org/GNOME/gtk	https://gitlab.gnome.org/GNOME/gtk/-/blob/main/COPYING
gts	https://sourceforge.net/projects/gts/	https://www.gnu.org/licenses/lgpl-3.0.html
h11	https://github.com/python-hyper/h11	https://github.com/python-hyper/h11/blob/master/LICENSE.txt
h2	https://github.com/python-hyper/h2	https://github.com/python-hyper/h2?tab=MIT-1-ov-file#readme

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
h5py	https://www.h5py.org	https://docs.h5py.org/en/stable/licenses.html
harfbuzz	https://github.com/harfbuzz/harfbuzz	https://github.com/harfbuzz/harfbuzz/blob/main/COPYING
hdbscan	https://hdbscan.readthedocs.io/en/latest/	https://github.com/scikit-learn-contrib/hdbscan/blob/master/LICENSE
hdf4	https://www.hdfgroup.org/solutions/hdf4/	https://github.com/HDFGroup/hdf4/blob/master/COPYING
hdf5	https://www.hdfgroup.org/solutions/hdf5/	https://github.com/HDFGroup/hdf5/blob/master/COPYING
he	https://github.com/mathiasbynens/he	https://github.com/mathiasbynens/he/raw/master/LICENSE-MIT.txt
hpack	https://github.com/python-hyper/hpack	https://github.com/python-hyper/hpack?tab=MIT-1-ov-file#readme
html-loader	https://github.com/webpack-contrib/html-loader	https://github.com/webpack-contrib/html-loader/raw/master/LICENSE
html5lib	https://github.com/html5lib/html5lib-python	https://github.com/html5lib/html5lib-python/blob/master/LICENSE
htslib	https://www.htslib.org	https://github.com/samtools/htslib/blob/develop/LICENSE
httpcore	https://www.encode.io/httpcore/	https://github.com/encode/httpcore/blob/master/LICENSE.md
httpx	https://github.com/encode/httpx	https://github.com/encode/httpx/blob/master/LICENSE.md

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
humanize	https://github.com/jmoiron/humanize	https://github.com/jmoiron/humanize/blob/master/LICENCE
hydra-core	https://github.com/facebookresearch/hydra	https://github.com/facebookresearch/hydra/blob/main/LICENSE
hyperframe	https://github.com/python-hyper/hyperframe	https://github.com/python-hyper/hyperframe?tab=MIT-1-ov-file#readme
icu	https://github.com/unicode-org/icu	https://github.com/unicode-org/icu/blob/main/icu4c/LICENSE
idna	https://github.com/kjd/idna	https://github.com/kjd/idna/blob/master/LICENSE.md
ihm	https://github.com/ihmwg/python-ihm	https://github.com/ihmwg/python-ihm/blob/main/LICENSE
imageio	https://github.com/imageio/imageio	https://github.com/imageio/imageio/blob/master/LICENSE
imagesize	https://github.com/shibukawa/imagesize_py	https://github.com/shibukawa/imagesize_py/blob/master/LICENSE.rst
ImmuneBuilder	https://github.com/oxpig/ImmuneBuilder/tree/main	https://github.com/oxpig/ImmuneBuilder/blob/main/LICENSE
importlib-metadata	https://github.com/python/importlib_metadata	https://github.com/python/importlib_metadata/blob/main/LICENSE
importlib_metadata	https://github.com/python/importlib_metadata	https://github.com/python/importlib_metadata/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
importlib_resources	https://github.com/python/importlib_resources	https://github.com/python/importlib_resources/blob/main/LICENSE
InChI	https://iupac.org/who-we-are/divisions/division-details/inchi/	https://iupac.org/wp-content/uploads/2021/06/iupac-inchi-license_2020.pdf
inflection	https://github.com/jinzhu/inflection	https://github.com/jinzhu/inflection/blob/master/LICENSE
ini.v1	https://gopkg.in/ini.v1	https://github.com/go-ini/ini/blob/v1.66.2/LICENSE
iniconfig	https://github.com/pytest-dev/iniconfig	https://github.com/pytest-dev/iniconfig/blob/main/LICENSE
innersvg-polyfill	https://github.com/dnozay/innersvg-polyfill	https://github.com/dnozay/innersvg-polyfill/raw/master/LICENSE
intel-openmp	https://github.com/hermitcore/libomp_oss	https://github.com/hermitcore/libomp_oss/blob/hermit/LICENSE.txt
ipykernel	https://ipython.org	https://ipython.org
ipython	https://ipython.org	https://github.com/ipython/ipython/blob/master/LICENSE
ipython-genutils	http://ipython.org	https://github.com/ipython/ipython_genutils/blob/master/COPYING.md
ipython_pygments_le	https://github.com/ipython/ipython-pygments-lex	https://github.com/ipython/ipython-pygments-lex blob/main/LICENSE
ipywidgets	http://jupyter.org	https://github.com/jupyter-widgets/ipywidgets/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
isodate	https://github.com/gweis/isodate/	https://github.com/gweis/isodate/blob/master/LICENSE
isoduration	https://github.com/bolsote/isoduration	https://github.com/bolsote/isoduration?tab=ISC-1-ov-file#readme
itsdangerous	https://palletsprojects.com/p/itsdangerous/	https://itsdangerous.palletsprojects.com/en/2.1.x/license/
jax	https://github.com/google/jax	https://github.com/google/jax/blob/main/LICENSE
jaxlib	https://github.com/google/jax	https://github.com/google/jax/blob/main/LICENSE
jaxtyping	https://github.com/patrick-kidger/jaxtyping	https://github.com/patrick-kidger/jaxtyping?tab=License-1-ov-file#readme
jedi	https://jedi.readthedocs.io/en/latest/	https://github.com/davidhalter/jedi/blob/master/LICENSE.txt
Jinja2	https://palletsprojects.com/p/jinja/	https://github.com/pallets/jinja/blob/main/LICENSE.rst
jmespath	https://github.com/jmespath/jmespath.py	https://github.com/jmespath/jmespath.py/blob/develop/LICENSE.txt
joblib	https://joblib.readthedocs.io/en/latest/	https://github.com/joblib/joblib/blob/master/LICENSE.txt
jpeg	https://www.ijg.org	https://www.ijg.org/files/README
json5	https://github.com/dpranke/pyjson5	https://github.com/dpranke/pyjson5/blob/main/LICENSE
jsonfield	https://github.com/dmkoch/django-jsonfield/	https://github.com/rpkilby/jsonfield/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
jsonpatch	https://github.com/stefankoegl/python-json-patch	https://github.com/stefankoegl/python-json-patch/blob/master/LICENSE
jsonpickle	https://github.com/jsonpickle/jsonpickle	https://github.com/jsonpickle/jsonpickle/blob/main/LICENSE
jsonpointer	https://github.com/stefankoegl/python-json-pointer	https://github.com/stefankoegl/python-json-pointer/blob/master/LICENSE.txt
jsonschema	https://github.com/python-jsonschema/jsonschema	https://github.com/python-jsonschema/jsonschema/blob/master/COPYING
jsonschema-specifications	https://github.com/python-jsonschema/jsonschema-specificat blob/main/README.rst	https://github.com/python-jsonschema/jsonschema-specificat blob/main/COPYING
jstat	https://github.com/jstat/jstat	https://www.opensource.org/licenses/mit-license.php
jupyter-client	https://jupyter.org	https://github.com/jupyter/jupyter_client/blob/main/COPYING.md
jupyter-core	https://jupyter.org	https://github.com/jupyter/jupyter_core/blob/master/COPYING.md
jupyter-events	https://github.com/jupyter/jupyter_events	https://github.com/jupyter/jupyter_events/blob/main/LICENSE
jupyter-lsp	https://github.com/jupyter-lsp/jupyterlab-lsp	https://github.com/jupyter-lsp/jupyterlab-lsp
jupyter-serverhttp://jupyter.or	https://github.com/jupyter-server/jupyter_server/blob/main/LICENSE	
jupyter_client	http://jupyter.org	https://github.com/jupyter/jupyter_client/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
jupyter_core	http://jupyter.org	https://github.com/jupyter/jupyter_core/blob/master/LICENSE
jupyter_server	https://github.com/jupyter-server/jupyter_server	https://github.com/jupyter-server/jupyter_server/blob/main/LICENSE
jupyter_server_termin	https://github.com/jupyter-server/jupyter_server_terminals	https://github.com/jupyter-server/jupyter_server_terminals/blob/main/LICENSE
jupyterlab	https://github.com/jupyterlab/jupyterlab	https://github.com/jupyterlab/jupyterlab/blob/main/LICENSE
jupyterlab-pygments	http://jupyter.org	https://github.com/jupyterlab/jupyterlab_pygments/blob/main/LICENSE
jupyterlab-widgets	http://jupyter.org	https://github.com/jupyter-widgets/ipywidgets/blob/master/LICENSE
jupyter-lab_pygments	https://github.com/jupyterlab/jupyterlab_pygments	https://github.com/jupyterlab/jupyterlab_pygments?tab=BSD-3-Clause-1-ov-file#readme
jupyterlab_server	https://github.com/jupyterlab/jupyterlab_server	https://github.com/jupyterlab/jupyterlab_server/blob/main/LICENSE
jupyterlab_widgets	https://github.com/jupyter-widgets/ipywidgets	https://github.com/jupyter-widgets/ipywidgets?tab=BSD-3-Clause-1-ov-file#readme
kaleido	https://github.com/plotly/Kaleido	https://github.com/plotly/Kaleido/blob/master/LICENSE.txt
keras	https://github.com/keras-team/keras	https://github.com/keras-team/keras/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
Keras-Preprocessing	https://github.com/keras-team/keras-preprocessing	https://github.com/keras-team/keras-preprocessing/blob/master/LICENSE
keras-tuner	https://github.com/keras-team/keras-tuner	https://github.com/keras-team/keras-tuner/blob/master/LICENSE
keyring	https://github.com/jaraco/keyring	https://github.com/jaraco/keyring/blob/main/LICENSE
keyutils	https://github.com/sassoftware/python-keyutils	https://github.com/sassoftware/python-keyutils/blob/master/LICENSE
kiwisolver	https://kiwisolver.readthedocs.io/en/latest/	https://github.com/nucleic/kiwi/blob/main/LICENSE
kombu	https://kombu.readthedocs.io	https://github.com/celery/kombu/blob/master/LICENSE
krb5	https://web.mit.edu/kerberos/	https://github.com/krb5/krb5/blob/master/src/lib/gssapi/LICENSE
kt-legacy	https://github.com/haifeng-jin/kt-legacy	https://github.com/haifeng-jin/kt-legacy/blob/main/LICENSE
lark	https://github.com/lark-parser/lark	https://github.com/lark-parser/lark?tab=MIT-1-ov-file#readme
lazy_loader	https://github.com/scientific-python/lazy_loader	https://github.com/scientific-python/lazy_loader/blob/main/LICENSE.md
lcms2	https://www.littlecms.com	https://github.com/sk1project/python-lcms2/blob/master/LICENSE
legacy-cgi	https://github.com/jackrosenthal/legacy-cgi	https://github.com/jackrosenthal/legacy-cgi?tab=License-1-ov-file#readme

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
lerc	https://github.com/Esri/lerc	https://github.com/Esri/lerc/blob/master/LICENSE
libarchive	http://www.libarchive.org	https://raw.githubusercontent.com/libarchive/libarchive/master/COPYING
libblas	https://www.netlib.org/blas/	https://www.netlib.org/blas/#_licensing
libbrotlicommon	https://github.com/google/brotli	https://github.com/google/brotli/blob/master/LICENSE
libbrotlidec	https://github.com/google/brotli	https://github.com/google/brotli/blob/master/LICENSE
libbrotlienc	https://github.com/google/brotli	https://github.com/google/brotli/blob/master/LICENSE
libcblas	https://anaconda.org/conda-forge/libcblas	N/A
libclang	https://github.com/sighingnow/libclang	https://github.com/sighingnow/libclang/blob/master/LICENSE.TXT
libcurl	https://curl.se/libcurl/	https://curl.se/docs/copyright.html
libcxx	https://github.com/llvm-mirror/libcxx	https://github.com/llvm-mirror/libcxx/blob/master/LICENSE.TXT
libdb	https://www.oracle.com/technology/software/products/berkeley-db/index.html	https://github.com/berkeleydb/libdb/blob/master/LEGAL
libdeflate	https://github.com/ebiggers/libdeflate	https://github.com/ebiggers/libdeflate/blob/master/COPYING
libedit	https://thrysoee.dk/editline/	http://www.netbsd.org/Goals/redistribution.html
libev	https://software.schmorp.de/pkg/libev.html	https://software.schmorp.de/pkg/libev.html

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
libexpat	https://libexpat.github.io/	https://github.com/libexpat/libexpat?tab=MIT-1-ov-file#readme
libffi	https://github.com/libffi/libffi	https://github.com/libffi/libffi/blob/master/LICENSE
libgcc	https://gcc.gnu.org/onlinedocs/gccint/Libgcc.html	https://gcc.gnu.org/onlinedocs/libstdc++/manual/license.html
libgcrypt	https://gnupg.org/software/index.html	https://git.gnupg.org/cgi-bin/gitweb.cgi?p=libgcrypt.git;a=blob;f=COPYING.LIB;hb=HEAD
libgd	https://libgd.github.io	https://libgd.github.io/manuals/2.2.1/files/license-txt.html
libglib	https://github.com/PolMine/libglib	https://github.com/PolMine/libglib/blob/master/include/glib-2.0/glib.h
libiconv	https://www.gnu.org/software/libiconv/	https://www.gnu.org/software/libiconv/#downloading
libint	https://tinyurl.com/yvw97wbw	https://tinyurl.com/yvw97wbw
liblapack	http://www.netlib.org/lapack/	https://www.netlib.org/lapack/LICENSE.txt
liblapacke	https://anaconda.org/conda-forge/liblapacke	https://www.netlib.org/lapack/LICENSE.txt
liblzma	https://tukaani.org/xz/	https://git.tukaani.org/?p=xz.git;a=blob;f=COPYING.0BSD
libmamba	https://medium.com/@QuantStack/open-software-packag	https://github.com/mamba-org/mamba/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
libmambapy	https://www.anaconda.com/blog/a-faster-conda-for-a-g	https://github.com/conda/conda-libmamba-solv blob/main/LICENSE
libnetcdf	https://www.unidata.ucar.edu/software/netcdf/	https://www.unidata.ucar.edu/software/netcdf/copyright.html
libnghttp2	https://github.com/nghttp2/nghttp2	https://github.com/nghttp2/nghttp2/blob/master/COPYING
libnsl	https://github.com/thkukuk/libnsl	https://gcc.gnu.org/onlinedocs/libstdc++/manual/license.html
libopenblas	https://www.openblas.net/	https://github.com/xianyi/OpenBLAS/blob/develop/LICENSE
libpng	https://www.libpng.org/pub/png/libpng.html	https://www.libpng.org/pub/png/src/libpng-LICENSE.txt
libpq	https://www.postgresql.org/	https://www.postgresql.org/about/licence/
librsvg	https://gitlab.gnome.org/GNOME/librsvg	https://gitlab.gnome.org/GNOME/librsvg/-/blob/main/COPYING.LIB
libsolv	https://github.com/openSUSE/libsolv	https://github.com/openSUSE/libsolv/blob/master/LICENSE.BSD
libsqlite	https://github.com/mackyle/sqlite	https://github.com/mackyle/sqlite?tab=License-1-ov-file#readme
libssh2	https://github.com/libssh2/libssh2	https://github.com/libssh2/libssh2/blob/master/COPYING
libstdcxx	https://gcc.gnu.org/	https://gcc.gnu.org/onlinedocs/libstdc++/manual/license.html

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
libtiff	https://www.libtiff.org/	https://www.libtiff.org/misc.html
libtrust	https://github.com/docker/libtrust	https://github.com/docker/libtrust/blob/master/LICENSE
libuuid	https://sourceforge.net/projects/libuuid/	https://sourceforge.net/p/libuuid/code/ci/master/tree/COPYING
libuv	https://github.com/libuv/libuv	https://github.com/libuv/libuv/blob/v1.x/LICENSE
libwebp	https://chromium.googlesource.com/?format=HTML	https://chromium.googlesource.com/chromium/src/third_party/+/refs/heads/main/libwebp/LICENSE
libwebp-base	https://chromium.googlesource.com/?format=HTML	https://chromium.googlesource.com/chromium/src/third_party/+/refs/heads/main/libwebp/LICENSE
libxc	https://www.tddft.org/programs/libxc/	https://gitlab.com/libxc/libxc/-/blob/master/COPYING
libxcb	https://xcb.freedesktop.org	https://github.com/freedesktop/xcb-libxcb/blob/master/COPYING
libxcrypt	https://github.com/besser82/libxcrypt	https://github.com/besser82/libxcrypt?tab=LGPL-2.1-1-ov-file#readme
libxml2	https://git.gnome.org/browse/libxml2/	https://gitlab.gnome.org/GNOME/libxml2/-/blob/master/Copyright
libxmlsec1	https://github.com/lsh123/xmlsec	https://github.com/lsh123/xmlsec/blob/master/README
libxslt	https://gitlab.gnome.org/GNOME/libxslt/-/wikis/home	https://gitlab.gnome.org/GNOME/libxslt/-/blob/master/Copyright

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
libzlib	https://zlib.net	https://zlib.net/zlib_license.html
lightning-utilities	https://github.com/Lightning-AI/utilities	https://github.com/Lightning-AI/utilities?tab=Apache-2.0-1-ov-file#readme
lime	https://github.com/marcotcr/lime	https://github.com/marcotcr/lime/blob/master/LICENSE
linear-operator	https://github.com/cornellius-gp/linear_operator	https://github.com/cornellius-gp/linear_operator?tab=MIT-1-ov-file#readme
lit	http://llvm.org	https://releases.llvm.org/14.0.0/LICENSE.TXT
llvm-openmp	https://github.com/llvm-mirror/openmp	https://github.com/llvm-mirror/openmp/blob/master/LICENSE.txt
llvmlite	http://llvmlite.readthedocs.io	https://github.com/numba/llvmlite/blob/main/LICENSE
loader-utils	https://github.com/webpack/loader-utils	https://github.com/webpack/loader-utils/raw/master/LICENSE
logistro	https://github.com/geopozo/logistro	https://github.com/geopozo/logistro?tab=MIT-1-ov-file#readme
logomaker	https://logomaker.readthedocs.io/en/latest/	https://github.com/jbkinney/logomaker/blob/master/LICENSE
logrus	https://github.com/sirupsen/logrus	https://github.com/sirupsen/logrus/blob/master/LICENSE
logrus-airbrake-hook.v2	https://gopkg.in/gemnasium/logrus-airbrake-hook.v2	https://github.com/gemnasium/logrus-airbrake-hook/blob/v2.1.2/LICENSE
lxml	https://lxml.de	https://lxml.de/index.html#license

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
lz4-c	https://www.lz4.org/	https://github.com/lz4/lz4/blob/dev/LICENSE
lzo	https://www.oberhumer.com/opensource/lzo/	https://www.oberhumer.com/opensource/gpl.html
mamba	https://github.com/mamba-org/mamba	https://github.com/mamba-org/mamba?tab=BSD-3-Clause-1-ov-f readme
markdown	https://github.com/evilstreak/markdown-js	https://www.opensource.org/licenses/mit-license.php
markdown-it-py	https://github.com/executablebooks/markdown-it-py	https://github.com/executablebooks/markdown-it-py/blob/master/LICENSE
markupsafe	https://github.com/pallets/markupsafe/tree/main/src/markupsafe	https://github.com/pallets/markupsafe/tree/main?tab=BSD-3-Clause-1-ov-f
MarkupSafe	https://palletsprojects.com/p/markupsafe/	https://github.com/pallets/markupsafe/blob/main/LICENSE.rst
mashumaro	https://github.com/Fatal1ty/mashumaro	https://github.com/Fatal1ty/mashumaro
matplotlib	https://matplotlib.org	https://matplotlib.org/stable/users/project/license.html
matplotlib-base	https://matplotlib.org	https://matplotlib.org/stable/users/project/license.html
matplotlib-inline	https://github.com/ipython/matplotlib-inline	https://github.com/ipython/matplotlib-inline/blob/master/LICENSE
mda-xdrlib	https://github.com/MDAnalysis/mda-xdrlib	https://github.com/MDAnalysis/mda-xdrlib/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
mda_xdrlib	https://github.com/MDAnalysis/mda-xdrlib	https://github.com/MDAnalysis/mda-xdrlib?tab=License-1-ov-file#readme
mdtraj	https://www.mdtraj.org/	https://www.mdtraj.org/1.9.8.dev0/index.html#license
mdurl	https://github.com/markdown-it/mdurl	https://github.com/markdown-it/mdurl/blob/master/LICENSE
menuinst	https://github.com/conda/menuinst	https://github.com/conda/menuinst/blob/main/LICENSE.txt
mergo	https://github.com/imdario/mergo	https://github.com/imdario/mergo/blob/master/LICENSE
mistune	https://github.com/lepture/mistune	https://github.com/lepture/mistune/blob/master/LICENSE
mkl	https://github.com/rust-math/intel-mkl-src	https://github.com/rust-math/intel-mkl-src/blob/master/License.txt
mkl-fft	https://github.com/IntelPython/mkl_fft	https://github.com/IntelPython/mkl_fft/blob/master/LICENSE.txt
mkl-random	https://github.com/IntelPython/mkl_random	https://github.com/IntelPython/mkl_random/blob/master/LICENSE.txt
mkl-service	https://github.com/IntelPython/mkl-service	https://github.com/IntelPython/mkl-service/blob/master/LICENSE.txt
mkl_fft	http://github.com/IntelPython/mkl_fft	https://github.com/IntelPython/mkl_fft?tab=BSD-3-Clause-1-ov-f readme

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
mkl_random	http://github.com/IntelPython/mkl_random	https://github.com/IntelPython/mkl_random?tab=BSD-3-Clause-1-ov-f readme
ml-dtypes	https://github.com/jax-ml/ml_dtypes	https://github.com/jax-ml/ml_dtypes/blob/main/LICENSE
ml_dtypes	https://github.com/jax-ml/ml_dtypes	https://github.com/jax-ml/ml_dtypes/blob/main/LICENSE
modelcif	https://github.com/ihmwg/python-modelcif	https://github.com/ihmwg/python-modelcif/blob/main/LICENSE
molecupy	https://molecupy.readthedocs.io/en/latest/	https://github.com/samirelanduk/atomium/blob/master/LICENSE
moment	https://github.com/moment/moment	https://github.com/moment/moment/raw/master/LICENSE
moment-precise-range-plugin	https://github.com/codebox/moment-precise-range	https://github.com/codebox/moment-precise-range raw/master/LICENSE.md
more-itertools	https://github.com/more-itertools/more-itertools	https://github.com/more-itertools/more-itertools/blob/master/LICENSE
mpiplus	https://github.com/choderalab/mpiplus	https://github.com/choderalab/mpiplus/blob/main/LICENSE
mpmath	http://mpmath.org/	https://github.com/mpmath/mpmath/blob/master/LICENSE
mrcfile	https://github.com/ccpem/mrcfile	https://github.com/ccpem/mrcfile/blob/master/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
msgpack	https://msgpack.org/	https://github.com/msgpack/msgpack-python/blob/main/COPYING
multidict	https://github.com/aio-libs/multidict	https://github.com/aio-libs/multidict/blob/master/LICENSE
multierr	https://go.uber.org/multierr	https://github.com/uber-go/multierr/blob/master/LICENSE.txt
multiprocess	https://github.com/uqfoundation/multiprocess	https://github.com/uqfoundation/multiprocess/blob/master/LICENSE
munkres	https://software.clapper.org/munkres/	https://github.com/bmc/munkres/blob/master/LICENSE.md
myesui uuid	https://github.com/myesui/uuid	https://pkg.go.dev/github.com/myesui/uuid?tab=licenses
namex	https://github.com/bcgov/namex	https://github.com/bcgov/namex/blob/main/LICENSE
narwhals	https://github.com/narwhals-dev/narwhals	https://github.com/narwhals-dev/narwhals?tab=MIT-1-ov-file#readme
nbclassic	http://jupyter.org	https://github.com/jupyter/nbclassic/blob/main/LICENSE
nbclient	https://jupyter.org	https://github.com/jupyter/nbclient/blob/main/LICENSE
nbconvert	https://jupyter.org	https://github.com/jupyter/nbconvert/blob/main/LICENSE
nbformat	http://jupyter.org	https://github.com/jupyter/nbformat/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
ncurses	https://invisible-island.net/ncurses/	https://invisible-island.net/ncurses/ncurses-license.html
ndfes	https://rutgerslbsr.gitlab.io/fe-toolkit/	https://gitlab.com/RutgersLBSR/fe-toolkit/-/blob/master/LICENSE
ndindex	https://quansight-labs.github.io/ndindex/	https://github.com/Quansight-Labs/ndindex?tab=MIT-1-ov-file#readme
nest-asyncio	https://github.com/erdewit/nest_asyncio	https://github.com/erdewit/nest_asyncio/blob/master/LICENSE
nest_asyncio	https://github.com/erdewit/nest_asyncio	https://github.com/erdewit/nest_asyncio?tab=BSD-2-Clause-1-ov-f readme
netcdf-fortran	https://www.unidata.ucar.edu/software/netcdf/	https://www.unidata.ucar.edu/software/netcdf/copyright.html
netCDF4	http://github.com/Unidata/netcdf4-python	https://github.com/Unidata/netcdf4-python/blob/master/LICENSE
nettle	https://git.lysator.liu.se/nettle/nettle	https://git.lysator.liu.se/nettle/nettle/-/blob/master/AUTHORS?ref_type=heads
networkx	https://networkx.org	https://github.com/networkx/networkx/blob/main/LICENSE.txt
nfpm	https://github.com/goreleaser/nfpm	https://github.com/goreleaser/nfpm/blob/master/LICENSE.md
ng-tags-input	https://github.com/mbenford/ngTagsInput	https://github.com/mbenford/ngTagsInput
ng-toast	https://github.com/tameraydin/ngToast	https://github.com/tameraydin/ngToast

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
ngdraggable	https://github.com/fatlinesofcode/ngDraggable	https://github.com/fatlinesofcode/ngDraggable/raw/master/LICENSE.txt
ngVue	https://github.com/ngVue/ngVue	https://github.com/ngVue/ngVue/raw/master/LICENSE
nh3	https://nh3.readthedocs.io/en/latest/	https://github.com/messense/nh3/blob/main/LICENSE
nlohmann_json	https://github.com/nlohmann/json	https://github.com/nlohmann/json?tab=MIT-1-ov-file#readme
nlopt	https://nlopt.readthedocs.io/en/latest/NLopt_Python_Reference/	https://nlopt.readthedocs.io/en/latest/NLopt_License_and_Copyright/
nodejs	https://nodejs.org/en/	https://raw.githubusercontent.com/nodejs/node/master/LICENSE
nomkl	https://github.com/conda-forge/nomkl-feedstock	https://github.com/conda-forge/nomkl-feedstock/blob/main/LICENSE.txt
notebook	http://jupyter.org	https://github.com/jupyter/notebook/blob/main/LICENSE
notebook-shim	https://github.com/jupyter/notebook_shim	https://github.com/jupyter/notebook_shim
notebook_shim	http://jupyter.org	https://github.com/jupyter/notebook_shim/blob/main/LICENSE
numba	https://numba.pydata.org	https://github.com/numba/numba/blob/main/LICENSE
numcpus	https://github.com/tklauser/numcpus	https://github.com/tklauser/numcpus/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
numexpr	https://github.com/pydata/numexpr	https://github.com/pydata/numexpr/blob/master/LICENSE.txt
numpy	https://numpy.org	https://github.com/numpy/numpy/blob/main/LICENSE.txt
numpy-base	https://numpy.org	https://github.com/numpy/numpy/blob/main/LICENSE.txt
numpydoc	https://numpydoc.readthedocs.io/en/latest/index.html	https://github.com/numpy/numpydoc/blob/main/LICENSE.txt
nvidia-cublas-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cublas-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cuda-cupti-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cuda-cupti-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cuda-nvrtc-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cuda-nvrtc-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cuda-runtime-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cuda-runtime-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cudnn-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/deeplearning/cudnn/sla/index.html
nvidia-cudnn-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/deeplearning/cudnn/sla/index.html

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
nvidia-cuEquivariance	https://github.com/NVIDIA/cuEquivariance	https://github.com/NVIDIA/cuEquivariance/blob/main/LICENSES/LICENSE
nvidia-cufft-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cufft-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cufile-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html#
nvidia-curand-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-curand-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cusolver-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cusolver-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cusparse-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cusparse-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-cusparselt-cu12	https://docs.nvidia.com/cuda/cusparselt/index.html	https://docs.nvidia.com/cuda/cusparselt/license.html
nvidia-ml-py	https://pypi.org/project/nvidia-ml-py/	https://pypi.org/project/nvidia-ml-py/
nvidia-nccl-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/hpc-sdk/eula/index.html
nvidia-nccl-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/hpc-sdk/eula/index.html
nvidia-nvjitlink-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
nvidia-nvshmem-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-nvtx-cu11	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
nvidia-nvtx-cu12	https://developer.nvidia.com/cuda-zone	https://docs.nvidia.com/cuda/eula/index.html
Oat++	https://oatpp.io/	https://github.com/oatpp/oatpp/blob/master/LICENSE
oauthlib	https://github.com/oauthlib/oauthlib	https://github.com/oauthlib/oauthlib/blob/master/LICENSE
ocl-icd	https://github.com/OCL-dev/ocl-icd	https://github.com/OCL-dev/ocl-icd/blob/master/COPYING
ocl-icd-system	https://github.com/conda-forge/ocl-icd-system-feedst	https://github.com/conda-forge/ocl-icd-system-feedst blob/main/LICENSE.txt
olefile	https://www.decalage.info/python/olefileio	https://olefile.readthedocs.io/en/latest/License.html
omegaconf	https://github.com/omry/omegaconf	https://github.com/omry/omegaconf/blob/master/LICENSE
OmegaFold	https://github.com/HeliXonProtein/OmegaFold/tree/main	https://github.com/HeliXonProtein/OmegaFold/blob/main/LICENSE
omnicanvas	https://omnicanvas.readthedocs.io/en/latest/	https://github.com/samirelanduk/omnicanvas/blob/master/LICENSE
OpenFF	https://openforcefield.org/	https://github.com/openforcefield/openff-forcefields
openff-amber-ff-ports	https://github.com/openforcefield/openff-amber-ff-ports	https://github.com/openforcefield/openff-amber-ff-ports blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
openff-forcefields	https://openforcefield.org	https://github.com/openforcefield/openff-forcefields/blob/master/LICENSE
openff-fragmenter	https://github.com/openforcefield/openff-fragmenter	https://github.com/openforcefield/openff-fragmenter/blob/main/LICENSE
openff-interchange	https://github.com/openforcefield/openff-interchange	https://github.com/openforcefield/openff-interchange/blob/main/LICENSE
openff-models	https://github.com/openforcefield/openff-models	https://github.com/openforcefield/openff-models/blob/main/LICENSE
openff-nagl	https://docs.openforcefield.org/projects/nagl/en/latest/?badge=latest	https://github.com/openforcefield/openff-nagl/blob/main/LICENSE
openff-nagl-models	https://github.com/openforcefield/openff-nagl-models	https://github.com/openforcefield/openff-nagl-models/blob/main/LICENSE
openff-toolkit	https://openforcefield.org	https://github.com/openforcefield/openff-toolkit/blob/master/LICENSE
openff-toolkit-base	https://openforcefield.org	https://github.com/openforcefield/openff-toolkit/blob/master/LICENSE
openff-units	https://github.com/openforcefield/openff-units	https://github.com/openforcefield/openff-units/blob/main/LICENSE
openff-utilities	https://github.com/openforcefield/openff-utilities	https://github.com/openforcefield/openff-utilities/blob/main/LICENSE
openjpeg	https://github.com/uclouvain/openjpeg	https://github.com/uclouvain/openjpeg/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
openldap	https://www.openldap.org/software/repo.html	https://git.openldap.org/openldap/openldap/-/blob/master/LICENSE
OpenMM	https://openmm.org	https://docs.openmm.org/7.1.0/userguide/library.html#license
openmmtools	https://github.com/choderalab/openmmtools	https://github.com/choderalab/openmmtools/blob/main/LICENSE
openmoltools	https://github.com/choderalab/openmoltools	https://github.com/choderalab/openmoltools/blob/master/LICENSE
openpyxl	https://openpyxl.readthedocs.io/en/stable/	https://foss.heptapod.net/openpyxl/openpyxl/-/blob/branch/3.0/LICENCE.rst
openssl	https://www.openssl.org	https://github.com/openssl/openssl/blob/master/LICENSE.txt
opt-einsum	https://github.com/dgasmith/opt_einsum	https://github.com/dgasmith/opt_einsum/blob/master/LICENSE
opt_einsum	https://github.com/dgasmith/opt_einsum	https://github.com/dgasmith/opt_einsum/blob/main/LICENSE
OptKing	https://github.com/psi-rking/optking	https://github.com/psi-rking/optking/blob/master/LICENSE
optree	https://optree.readthedocs.io/en/latest/	https://github.com/metaopt/optree/blob/main/LICENSE
oscrypto	https://github.com/wbond/oscrypto	https://github.com/wbond/oscrypto/blob/master/LICENSE
overrides	https://github.com/mkorpela/overrides	https://github.com/mkorpela/overrides/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
packaging	https://github.com/pypa/packaging	https://github.com/pypa/packaging/blob/main/LICENSE
packmol	https://leandro.iqm.unicamp.br/m3g/packmol/home.shtml	https://github.com/m3g/packmol/blob/master/LICENSE
packmol_memgen	http://cpclab.uni-duesseldorf.de/	https://gitlab.com/cbjh/memgen/py-memgen/-/blob/main/LICENSE
pandas	https://pandas.pydata.org	https://github.com/pandas-dev/pandas/blob/main/LICENSE
pandocfilters	http://github.com/jgm/pandocfilters	https://github.com/jgm/pandocfilters/blob/master/LICENSE
panedr	https://github.com/MDAnalysis/panedr	https://github.com/MDAnalysis/panedr/blob/master/LICENSE.txt
pango	https://pango.gnome.org	https://gitlab.gnome.org/GNOME/pango/-/blob/main/COPYING
ParmEd	https://parmed.github.io/ParmEd/html/index.html	https://github.com/ParmEd/ParmEd#license
parser	https://github.com/typescript-eslint/typescript-eslint	https://github.com/typescript-eslint/typescript-eslint/raw/master/LICENSE
parso	https://parso.readthedocs.io/en/latest/	https://github.com/davidhalter/parso/blob/master/LICENSE.txt
pathos	https://github.com/uqfoundation/pathos	https://github.com/uqfoundation/pathos/blob/master/LICENSE
patsy	https://patsy.readthedocs.io/en/latest/	https://github.com/pydata/patsy/blob/master/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
pbkdf2	https://golang.org/x/crypto/pbkdf2	https://cs.opensource.google/go/x/crypto/+/master:LICENSE
pbr	https://docs.openstack.org/pbr/latest/	https://opendev.org/openstack/pbr/src/branch/master/LICENSE
pcmsolver	https://pcmsolver.readthedocs.io/en/stable/	https://github.com/PCMSolver/pcmsolver/blob/master/LICENSE
pcre	https://www.pcre.org	https://www.pcre.org/licence.txt
pcre2	https://www.pcre.org	https://github.com/PCRE2Project/pcre2/blob/master/LICENCE
pdb4amber	https://github.com/Amber-MD/pdb4amber	https://github.com/Amber-MD/pdb4amber/blob/master/LICENSE
pdbfixer	https://github.com/openmm/pdbfixer	https://github.com/openmm/pdbfixer/blob/master/LICENSE
pexpect	https://pexpect.readthedocs.io/	https://github.com/pexpect/pexpect/blob/master/LICENSE
pgconn	https://github.com/jackc/pgconn	https://github.com/jackc/pgconn/blob/master/LICENSE
pgio	https://github.com/jackc/pgio	https://github.com/jackc/pgio/blob/master/LICENSE
pgpassfile	https://github.com/jackc/pgpassfile	https://github.com/jackc/pgpassfile/blob/master/LICENSE
pgproto3	https://github.com/jackc/pgproto3/v2	https://github.com/jackc/pgproto3/blob/master/LICENSE
pgtype	https://github.com/jackc/pgtype	https://github.com/jackc/pgtype/blob/master/LICENSE
pgx	https://github.com/jackc/pgx/v4	https://github.com/jackc/pgx/blob/v4.0.1/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
phonopy	https://github.com/phonopy/phono3py	https://github.com/phonopy/phono3py/blob/develop/LICENSE
pickleshare	https://github.com/pickleshare/pickleshare	https://github.com/pickleshare/pickleshare/blob/master/LICENSE
Pillow	https://python-pillow.org	https://github.com/python-pillow/Pillow/blob/main/LICENSE
Pint	https://pint.readthedocs.io/en/stable/	https://github.com/hgrecco/pint/blob/master/LICENSE
pip	https://pip.pypa.io/	https://github.com/pypa/pip/blob/main/LICENSE.txt
pip-licenses	https://github.com/raimon49/pip-licenses	https://github.com/raimon49/pip-licenses/blob/master/LICENSE
pixman	https://pixman.org	https://cgit.freedesktop.org/pixman/tree/COPYING
pkginfo	https://launchpad.net/pkginfo	https://bazaar.launchpad.net/~tseaver/pkginfo/trunk/view/head:/LICENSE.txt
pkgutil-resolve-name	https://github.com/graingert/pkgutil-resolve-name	https://github.com/graingert/pkgutil-resolve-name?tab=License-1-ov-file#readme
pkgutil_resolve_name	https://github.com/graingert/pkgutil-resolve-name	https://github.com/graingert/pkgutil-resolve-name?tab=License-1-ov-file#readme
platformdirs	https://github.com/platformdirs/platformdirs	https://github.com/platformdirs/platformdirs/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
plotly	https://plotly.com/python/	https://github.com/plotly/plotly.py/blob/master/LICENSE.txt
plotly-orca	https://github.com/plotly/orca	https://github.com/plotly/orca/blob/master/LICENSE
plotly.js	https://github.com/plotly/plotly.js	https://github.com/plotly/plotly.js/raw/master/LICENSE
pluggy	https://pluggy.readthedocs.io/en/stable/index.html	https://github.com/pytest-dev/pluggy/blob/main/LICENSE
pooch	https://github.com/fatiando/pooch	https://github.com/fatiando/pooch/blob/main/LICENSE.txt
portalocker	https://github.com/WoLpH/portalocker	https://github.com/wolph/portalocker/blob/develop/LICENSE
pox	https://github.com/uqfoundation/pox	https://github.com/uqfoundation/pox/blob/master/LICENSE
ppft	https://github.com/uqfoundation/ppft	https://github.com/uqfoundation/ppft/blob/master/LICENSE
pq	https://github.com/lib/pq	https://github.com/lib/pq/blob/master/LICENSE.md
ProDy	http://www.csb.pitt.edu/ProDy	https://github.com/prody/ProDy/blob/master/LICENSE.rst
prometheus-client	https://github.com/prometheus/client_python	https://github.com/prometheus/client_python/blob/master/LICENSE
prometheus_client	https://github.com/prometheus/client_python	https://github.com/prometheus/client_python/blob/master/LICENSE
prompt-toolkit	https://python-prompt-toolkit.readthedocs.io/en/stable/	https://github.com/prompt-toolkit/python-prompt-toolkit/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
prompt_toolkit	https://github.com/prompt-toolkit/python-prompt-toolki	https://github.com/prompt-toolkit/python-prompt-toolki tab=BSD-3-Clause-1-ov-f readme
propcache	https://github.com/aio-libs/propcache	https://github.com/aio-libs/propcache?tab=Apache-2.0-1-ov-file#readme
protobuf	https://google.golang.org/protobuf	https://github.com/protocolbuffers/protobuf-go/blob/master/LICENSE
psi4	https://psicode.org	https://github.com/psi4/psi4/blob/master/COPYING.LESSER
psutil	https://psutil.readthedocs.io/en/latest/	https://github.com/giampaolo/psutil/blob/master/LICENSE
psycopg2	https://psycopg.org/	https://github.com/psycopg/psycopg2/blob/master/LICENSE
PTable	https://github.com/kxxoling/PTable	https://github.com/kxxoling/PTable/blob/master/COPYING
pthread-stubs	https://xcb.freedesktop.org	https://github.com/freedesktop/xcb-pthread-stubs/blob/master/COPYING
ptyprocess	https://ptyprocess.readthedocs.io/en/latest/	https://github.com/pexpect/ptyprocess/blob/master/LICENSE
pure-eval	http://github.com/alexmojaki/pure_eval	http://github.com/alexmojaki/pure_eval/blob/master/LICENSE.txt
pure_eval	http://github.com/alexmojaki/pure_eval	https://github.com/alexmojaki/pure_eval?tab=MIT-1-ov-file#readme

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
py	https://py.readthedocs.io/en/latest/	https://github.com/pytest-dev/py/blob/master/LICENSE
py-cpuinfo	https://github.com/workhorsy/py-cpuinfo	https://github.com/workhorsy/py-cpuinfo/blob/master/LICENSE
pyasn1	https://github.com/etingof/pyasn1	https://github.com/etingof/pyasn1/blob/master/LICENSE.rst
pyasn1-modules	https://github.com/etingof/pyasn1-modules	https://github.com/etingof/pyasn1-modules/blob/master/LICENSE.txt
pyasn1_modules	https://pyasn1.readthedocs.io/en/latest/contents.html	https://github.com/pyasn1/pyasn1-modules?tab=BSD-2-Clause-1-ov-file#readme
pybind11	http://pybind11.readthedocs.io/en/stable/	https://github.com/pybind/pybind11/blob/master/LICENSE
pybind11-abi	https://github.com/pybind/pybind11	https://github.com/pybind/pybind11/blob/master/LICENSE
pybind11_global	http://pybind11.readthedocs.io/en/stable/	https://github.com/pybind/pybind11/blob/master/LICENSE
pycairo	https://pycairo.readthedocs.io/en/latest/index.html	https://github.com/pygobject/pycairo/blob/master/COPYING
pycosat	https://github.com/conda/pycosat	https://github.com/conda/pycosat/blob/main/LICENSE
pycparser	https://github.com/eliben/pycparser	https://github.com/eliben/pycparser/blob/master/LICENSE
pydantic	https://pydantic-docs.helpmanual.io	https://github.com/samuelcolvin/pydantic/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
pydantic-core	https://github.com/pydantic/pydantic-core	https://github.com/pydantic/pydantic-core
pyedr	https://github.com/MDAnalysis/panedr	https://github.com/MDAnalysis/panedr/blob/master/LICENSE.txt
pyEMMA	http://github.com/markovmodel/PyEMMA	https://github.com/markovmodel/PyEMMA/blob/devel/LICENSE.txt
Pygments	https://pygments.org	https://github.com/pygments/pygments/blob/master/LICENSE
pygments	https://pygments.org/	https://github.com/pygments/pygments?tab=BSD-2-Clause-1-ov-file#readme
pygraphviz	https://pygraphviz.github.io	https://github.com/pygraphviz/pygraphviz/blob/main/LICENSE
pygtop	https://pygtop.readthedocs.io/en/latest/	https://github.com/samirelanduk/pygtop/blob/master/LICENSE
pyHanko	https://github.com/MatthiasValvekens/pyHanko	https://github.com/MatthiasValvekens/pyHanko/blob/master/LICENSE
pyhanko-certvalidator	https://github.com/MatthiasValvekens/certvalidator	https://github.com/MatthiasValvekens/certvalidator/blob/master/LICENSE
PyJWT	https://github.com/jpadilla/pyjwt	https://github.com/jpadilla/pyjwt/blob/master/LICENSE
pymbar	https://pymbar.org	https://github.com/choderalab/pymbar/blob/master/LICENSE
pynvml	https://github.com/gpuopenanalytics/pynvml	https://github.com/gpuopenanalytics/pynvml/blob/master/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
pyOpenSSL	https://pyopenssl.org/	https://github.com/pyca/pyopenssl/blob/main/LICENSE
pyparsing	https://pyparsing-docs.readthedocs.io/en/latest/	https://github.com/pyparsing/pyparsing/blob/master/LICENSE
PyPDF3	https://github.com/sfneal/PyPDF3	https://github.com/sfneal/PyPDF3/blob/master/LICENSE
pyrsistent	http://github.com/tobgu/pyrsistent/	https://github.com/tobgu/pyrsistent/blob/master/LICENSE.mit
pysam	https://github.com/pysam-developers/pysam	https://github.com/pysam-developers/pysam/blob/master/COPYING
PySocks	https://github.com/Anorov/PySocks	https://github.com/Anorov/PySocks/blob/master/LICENSE
pytables	https://www.pytables.org	https://github.com/PyTables/PyTables/blob/master/LICENSE.txt
python	https://www.python.org/	https://docs.python.org/3/license.html
python-bidi	https://github.com/MeirKriheli/python-bidi	https://github.com/MeirKriheli/python-bidi/blob/master/COPYING
python-constraint	https://github.com/python-constraint/python-constraint	https://github.com/python-constraint/python-constraint/blob/master/LICENSE
python-dateutil	https://dateutil.readthedocs.io	https://github.com/dateutil/dateutil/blob/master/LICENSE
python-json-logger	http://github.com/madzak/python-json-logger	https://github.com/madzak/python-json-logger/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
python-pptx	https://github.com/scanny/python-pptx	https://github.com/scanny/python-pptx/blob/master/LICENSE
python-ldap	https://www.python-ldap.org/	https://github.com/python-ldap/python-ldap/blob/python-ldap-3.4.0/LICENCE
python3-saml	https://github.com/onelogin/python3-saml	https://github.com/onelogin/python3-saml/blob/master/LICENSE
python_abi	https://github.com/conda-forge/python_abi-feedstock	https://github.com/conda-forge/python_abi-feedstock/blob/main/LICENSE.txt
pytorch-lightning	https://github.com/Lightning-AI/pytorch-lightning	https://github.com/Lightning-AI/pytorch-lightning/blob/master/LICENSE
pytz	https://pythonhosted.org/pytz	https://git.launchpad.net/pytz/tree/LICENSE.txt
pytz-deprecation-shim	https://github.com/pganssle/pytz-deprecation-shim	https://github.com/pganssle/pytz-deprecation-shim/blob/master/LICENSE
PyWavelets	https://github.com/PyWavelets/pywt	https://github.com/PyWavelets/pywt/blob/master/LICENSE
PyYAML	https://pyyaml.org/	https://github.com/yaml/pyyaml/blob/master/LICENSE
pyyml	No longer available	No longer available
pyzmq	https://pyzmq.readthedocs.io/en/latest/	https://github.com/zeromq/pyzmq/blob/main/COPYING.BSD
qcelemental	https://github.com/MolSSI/QCElemental	https://github.com/MolSSI/QCElemental/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
qcengine	https://github.com/MolSSI/QCEngine	https://github.com/MolSSI/QCEngine/blob/master/LICENSE
qrcode	https://github.com/lincolnloop/python-qrcode	https://github.com/lincolnloop/python-qrcode/blob/main/LICENSE
ramda	https://github.com/ramda/ramda	https://github.com/ramda/ramda/raw/master/LICENSE.txt
rapidjson	https://rapidjson.org/	https://github.com/Tencent/rapidjson/blob/master/license.txt
rdkit	https://www.rdkit.org	https://github.com/rdkit/rdkit/blob/master/license.txt
re2	https://github.com/google/re2	https://github.com/google/re2/blob/main/LICENSE
readme-renderer	https://github.com/pypa/readme_renderer	https://github.com/pypa/readme_renderer/blob/main/LICENSE
redis-py	https://github.com/andymccurdy/redis-py	https://github.com/redis/redis-py/blob/master/LICENSE
referencing	https://github.com/python-jsonschema/referencing	https://github.com/python-jsonschema/referencing
regex	https://github.com/mrabarnett/mrab-regex	https://github.com/mrabarnett/mrab-regex/blob/hg/LICENSE.txt
reportlab	https://www.reportlab.com	https://www.reportlab.com/dev/docs/faq/#1.3
reproc	https://github.com/DaanDeMeyer/reproc	https://github.com/DaanDeMeyer/reproc/blob/main/LICENSE
reproc-cpp	https://github.com/DaanDeMeyer/reproc	https://github.com/DaanDeMeyer/reproc/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
requests	https://requests.readthedocs.io	https://github.com/psf/requests/blob/main/LICENSE
requests-oauthlib	https://github.com/requests/requests-oauthlib	https://github.com/requests/requests-oauthlib/blob/master/LICENSE
requests-toolbelt	https://toolbelt.readthedocs.org	https://github.com/requests/toolbelt/blob/master/LICENSE
resumable	https://github.com/stevvooe/resumable	https://github.com/stevvooe/resumable/blob/master/LICENSE
retrying	https://github.com/rholder/retrying	https://github.com/rholder/retrying/blob/master/LICENSE
rfc3339-validator	https://github.com/naimetti/rfc3339-validator	https://github.com/naimetti/rfc3339-validator/blob/master/LICENSE
rfc3339_validator	https://github.com/naimetti/rfc3339-validator	https://github.com/naimetti/rfc3339-validator?tab=MIT-1-ov-file#readme
rfc3986	https://rfc3986.readthedocs.io/en/latest/	https://github.com/python-hyper/rfc3986/blob/main/LICENSE
rfc3986-validator	https://github.com/naimetti/rfc3986-validator	https://github.com/naimetti/rfc3986-validator/blob/master/LICENSE
rfc3987-syntax	https://github.com/willynilly/rfc3987-syntax	https://github.com/willynilly/rfc3987-syntax?tab=MIT-1-ov-file#readme
rich	https://github.com/Textualize/rich	https://github.com/Textualize/rich/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
rich-argparse	https://github.com/hamdanal/rich-argparse	https://github.com/hamdanal/rich-argparse/blob/main/LICENSE
rlPyCairo	https://pypi.org/project/rlPyCairo/	https://hg.reportlab.com/hg-public/rlPyCairo/file/tip/rlPyCairo/LICENSE.txt
RocksDict	https://github.com/rocksdict/RocksDict	https://github.com/rocksdict/RocksDict?tab=MIT-1-ov-file#readme
roman-numerals	https://github.com/AA-Turner/roman-numerals	https://github.com/AA-Turner/roman-numerals?tab=License-1-ov-file#readme
roman-numerals-py	https://github.com/AA-Turner/roman-numerals	https://github.com/AA-Turner/roman-numerals?tab=License-1-ov-file#readme
rotary-embedding-torch	https://github.com/lucidrains/rotary-embedding-tor	https://github.com/lucidrains/rotary-embedding-tor blob/main/LICENSE
rpds-py	https://github.com/crate-py/rpds	https://github.com/crate-py/rpds
rpmpack	https://github.com/google/rpmpack	https://github.com/google/rpmpack/blob/master/LICENSE
rsa	https://stuvel.eu/rsa	https://github.com/sybrenstuvel/python-rsa/blob/main/LICENSE
ruamel-yaml	https://sourceforge.net/p/ruamel-yaml/code/ci/default/tree/	https://sourceforge.net/p/ruamel-yaml/code/ci/default/tree/LICENSE
ruamel.yaml.clib	https://sourceforge.net/p/ruamel-yaml-clib/code/ci/default/tree/	https://sourceforge.net/p/ruamel-yaml-clib/code/ci/default/tree/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
s3transfer	https://github.com/boto/s3transfer	https://github.com/boto/s3transfer/blob/develop/LICENSE.txt
sacrebleu	https://github.com/mjpost/sacrebleu	https://github.com/mjpost/sacrebleu/blob/master/LICENSE.txt
sapiens	https://github.com/Merck/Sapiens	https://github.com/Merck/Sapiens/blob/main/LICENSE
sasl	https://mellium.im/sasl	https://github.com/mellium/sasl/blob/main/LICENSE
scikit-gstat	https://mmaelicke.github.io/scikit-gstat/	https://github.com/mmaelicke/scikit-gstat/blob/main/LICENSE
scikit-image	https://scikit-image.org	https://scikit-image.org/docs/dev/license.html
scikit-learn	https://scikit-learn.org/stable/	https://github.com/scikit-learn/scikit-learn/blob/main/COPYING
scikit-learn-extra	https://github.com/scikit-learn-contrib/scikit-learn-extra	https://github.com/scikit-learn-contrib/scikit-learn-extra/blob/main/LICENSE
scikit-network	https://github.com/sknetwork-team/scikit-network	https://github.com/sknetwork-team/scikit-network?tab=License-1-ov-file#readme
scipy	https://scipy.org	https://github.com/scipy/scipy/blob/main/LICENSE.txt
seaborn	https://seaborn.pydata.org	https://github.com/mwaskom/seaborn/blob/master/LICENSE
seaborn-base	https://seaborn.pydata.org	https://github.com/mwaskom/seaborn/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
semver	https://github.com/Masterminds/semver/v3	https://github.com/Masterminds/semver/blob/v3.0.3/LICENSE.txt
Send2Trash	https://github.com/arsenetar/send2trash	https://github.com/arsenetar/send2trash/blob/master/LICENSE
sentry-sdk	https://github.com/getsentry/sentry-python	https://github.com/getsentry/sentry-python/blob/master/LICENSE
setproctitle	https://github.com/dvarrazzo/py-setproctitle	https://github.com/dvarrazzo/py-setproctitle/blob/master/LICENSE
setuptools	https://github.com/pypa/setuptools	https://github.com/pypa/setuptools/blob/main/LICENSE
setuptools-scm	https://github.com/pypa/setuptools_scm/	https://github.com/pypa/setuptools_scm/blob/main/LICENSE
sh	https://github.com/amoffat/sh	https://github.com/amoffat/sh/blob/develop/LICENSE.txt
shellingham	https://github.com/sarugaku/shellingham	https://github.com/sarugaku/shellingham/blob/master/LICENSE
simdjson	https://github.com/simdjson/simdjson/tree/master	https://github.com/simdjson/simdjson/blob/master/LICENSE
simint	https://www.bennyp.org/research/simint/	https://github.com/simint-chem/simint-generator/blob/master/LICENSE
simplejson	https://github.com/simplejson/simplejson	https://github.com/simplejson/simplejson/blob/master/LICENSE.txt
six	https://github.com/benjaminp/six	https://github.com/benjaminp/six/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
smirnoff99frosst	https://github.com/openforcefield/smirnoff99frosst	https://github.com/openforcefield/smirnoff99Frosst/blob/master/LICENSE
smmap	https://github.com/gitpython-developers/smmap	https://github.com/gitpython-developers/smmap/blob/master/LICENSE
snappy	https://github.com/google/snappy	https://github.com/google/snappy/blob/main/COPYING
sniffio	https://github.com/python-trio/sniffio	https://github.com/python-trio/sniffio/blob/master/LICENSE.MIT
snowballstemmer	https://github.com/snowballstem/snowball	https://github.com/snowballstem/snowball/blob/master/COPYING
soupsieve	https://github.com/facelessuser/soupsieve	https://github.com/facelessuser/soupsieve/blob/main/LICENSE.md
spdlog	https://github.com/gabime/spdlog	https://github.com/gabime/spdlog?tab=License-1-ov-file#readme
spglib	https://spglib.readthedocs.io/	https://github.com/spglib/spglib/blob/develop/COPYING
sphinx	https://github.com/sphinx-doc/sphinx	https://github.com/sphinx-doc/sphinx/blob/5.x/LICENSE
sphinxcontrib-applehelp	https://github.com/sphinx-doc/sphinxcontrib-applehe	https://github.com/sphinx-doc/sphinxcontrib-applehe blob/master/LICENSE
sphinxcontrib-devhelp	https://github.com/sphinx-doc/sphinxcontrib-devhelp	https://github.com/sphinx-doc/sphinxcontrib-devhelp blob/master/LICENSE
sphinxcontrib-htmlhelp	https://github.com/sphinx-doc/sphinxcontrib-htmlhel	https://github.com/sphinx-doc/sphinxcontrib-htmlhel blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
sphinxcontrib-jsmath	https://github.com/sphinx-doc/sphinxcontrib-jsmath	https://github.com/sphinx-doc/sphinxcontrib-jsmath/blob/master/LICENSE
sphinxcontrib-qthelp	https://github.com/sphinx-doc/sphinxcontrib-qthelp	https://github.com/sphinx-doc/sphinxcontrib-qthelp/blob/master/LICENSE
sphinxcontrib-serializinghtml	https://github.com/sphinx-doc/sphinxcontrib-serializ	https://github.com/sphinx-doc/sphinxcontrib-serializ blob/master/LICENSE
SQLAlchemy	https://www.sqlalchemy.org	https://docs.sqlalchemy.org/en/14/copyright.html
SQLAlchemy	https://www.sqlalchemy.org/	https://github.com/sqlalchemy/sqlalchemy?tab=MIT-1-ov-file#readme
sqlite	https://sqlite.org/index.html	https://sqlite.org/copyright.html
sqlparse	https://github.com/andialbrecht/sqlparse	https://github.com/andialbrecht/sqlparse/blob/master/LICENSE
stack-data	http://github.com/alexmojaki/stack_data	https://github.com/alexmojaki/stack_data/blob/master/LICENSE.txt
stack_data	http://github.com/alexmojaki/stack_data	https://github.com/alexmojaki/stack_data?tab=MIT-1-ov-file#readme
starfile	https://github.com/alisterburt/starfile	https://github.com/teamtomo/starfile/blob/main/LICENSE
statsmodels	https://github.com/statsmodels/statsmodels	https://github.com/statsmodels/statsmodels/blob/main/LICENSE.txt
structlog	https://www.structlog.org/	https://github.com/hynek/structlog/blob/main/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
streamlit	https://github.com/streamlit/streamlit	https://github.com/streamlit/streamlit/blob/develop/LICENSE
svglib	https://github.com/deeplook/svglib	https://github.com/deeplook/svglib/blob/master/LICENSE.txt
sympy	https://sympy.org	https://github.com/sympy/sympy/blob/master/LICENSE
tables	https://www.pytables.org/	https://github.com/PyTables/PyTables/blob/master/LICENSE.txt
tabulate	https://github.com/astanin/python-tabulate	https://github.com/astanin/python-tabulate/blob/master/LICENSE
tbb	https://github.com/oneapi-src/oneTBB	https://github.com/oneapi-src/oneTBB/blob/master/LICENSE.txt
tenacity	https://github.com/jd/tenacity	https://github.com/jd/tenacity/blob/master/LICENSE
tensorboard	https://github.com/tensorflow/tensorboard	https://github.com/tensorflow/tensorboard/blob/master/LICENSE
tensorboard-data-server	https://github.com/tensorflow/tensorboard	https://github.com/tensorflow/tensorboard/blob/master/LICENSE
tensorboard-plugin-wit	https://github.com/pair-code/what-if-tool	https://github.com/PAIR-code/what-if-tool/blob/master/LICENSE
tensorboardX	https://github.com/lanpa/tensorboardX	https://github.com/lanpa/tensorboardX/blob/master/LICENSE
tensorflow	https://github.com/tensorflow/tensorflow	https://github.com/tensorflow/tensorflow/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
tensorflow-estimator	https://github.com/tensorflow/estimator	https://github.com/tensorflow/estimator/blob/master/LICENSE
tensorflow-io-gcs-filesystem	https://github.com/tensorflow/io	https://github.com/tensorflow/io/blob/master/LICENSE
tensorflow-probability	https://github.com/tensorflow/probability	https://github.com/tensorflow/probability/blob/main/LICENSE
termcolor	https://github.com/hugovk/termcolor	https://github.com/hugovk/termcolor/blob/main/COPYING.txt
terminado	https://github.com/jupyter/terminado	https://github.com/jupyter/terminado/blob/main/LICENSE
testpath	https://github.com/jupyter/testpath	https://github.com/jupyter/testpath/blob/master/LICENSE
textangular	https://github.com/fraywing/textAngular	https://github.com/fraywing/textAngular
texttable	https://github.com/foutaise/texttable/	https://github.com/foutaise/texttable/?tab=MIT-1-ov-file#readme
tf_keras	https://github.com/keras-team/keras-io	https://github.com/keras-team/keras-io/blob/master/LICENSE
tf_keras	https://github.com/keras-team/keras-io	https://github.com/keras-team/keras-io/blob/master/LICENSE
threadpoolctl	https://github.com/joblib/threadpoolctl	https://github.com/joblib/threadpoolctl/blob/master/LICENSE
three	https://github.com/mrdoob/three.js	https://github.com/mrdoob/three.js/raw/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
tiffile	https://github.com/cgohlke/tiffile/	https://github.com/cgohlke/tiffile/blob/master/LICENSE
tinycss2	https://github.com/Kozea/tinycss2/	https://github.com/Kozea/tinycss2/blob/master/LICENSE
tinyxml2	https://github.com/leethomason/tinyxml2	https://github.com/leethomason/tinyxml2/blob/master/LICENSE.txt
tk	https://www.tcl.tk/	https://www.tcl.tk/software/tcltk/license.html
toml	https://github.com/toml-lang/toml	https://github.com/toml-lang/toml/blob/main/LICENSE
tomli	https://github.com/hukkin/tomli	https://github.com/hukkin/tomli/blob/master/LICENSE
toolz	https://github.com/pytoolz/toolz	https://github.com/pytoolz/toolz/blob/master/LICENSE.txt
torch	https://pytorch.org/	https://pypi.org/project/torch/#license
torch-geometric	https://pyg.org/	https://github.com/pyg-team/pytorch_geometric?tab=MIT-1-ov-file#readme
torcheval	https://github.com/pytorch/torcheval	https://github.com/pytorch/torcheval?tab=License-1-ov-file#readme
torchmetrics	https://github.com/Lightning-AI/torchmetrics	https://github.com/Lightning-AI/torchmetrics?tab=Apache-2.0-1-ov-file#readme
tornado	https://www.tornadoweb.org	https://github.com/tornadoweb/tornado/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL		
tqdm	https://github.com/tqdm/tqdm	https://github.com/tqdm/tqdm/blob/master/LICENCE		
tracy	https://github.com/gear-genomics/tracy	https://github.com/gear-genomics/tracy/blob/main/LICENSE		
traitlets	https://github.com/ipython/traitlets	https://github.com/ipython/traitlets/blob/main/COPYING.md		
triton	https://github.com/openai/triton/	https://github.com/openai/triton/blob/main/LICENSE		
truststore	https://github.com/sethmlarson/trus	https://github.com/sethmlarson/truststore/blob/main/LICENSE		
truststore	https://github.com/sethmlarson/truststore	https://github.com/sethmlarson/truststore/blob/main/LICENSEtruststore	https://github.com/sethmlarson/truststore	https://github.com/sethmlarson/truststore/blob/main/LICENSE
ts-jest	https://github.com/kulshekhar/ts-jest	https://github.com/kulshekhar/ts-jest/raw/master/LICENSE.md		
ts-loader	https://github.com/TypeStrong/ts-loader	https://github.com/TypeStrong/ts-loader/raw/master/LICENSE		
twine	https://github.com/pypa/twine	https://github.com/pypa/twine/blob/main/LICENSE		
twinj uuid	https://github.com/twinj/uuid	https://github.com/twinj/uuid/blob/master/LICENSE		
types	https://github.com/babel/babel	https://github.com/babel/babel/raw/master/LICENSE		
types-python-dateutil	https://github.com/python/typeshed	https://github.com/python/typeshed?tab=License-1-ov-file#readme		
types-requests	https://pypi.org/project/types-requests/	https://github.com/python/typeshed/blob/main/LICENSE		

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
typescript	https://github.com/Microsoft/TypeScript	https://github.com/Microsoft/TypeScript/raw/master/LICENSE.txt
typing-inspection	https://github.com/pydantic/typing-inspection	https://github.com/pydantic/typing-inspection?tab=MIT-1-ov-file#readme
typing-utils	https://github.com/bojiang/typing_utils	https://github.com/bojiang/typing_utils?tab=Apache-2.0-1-ov-file#readme
typing_extensions	https://github.com/python/typing	https://github.com/python/typing/blob/master/LICENSE
typing_utils	https://github.com/bojiang/typing_utils	https://github.com/bojiang/typing_utils?tab=Apache-2.0-1-ov-file#readme
tzdata	https://github.com/python/tzdata	https://github.com/python/tzdata/blob/master/LICENSE
tzlocal	https://github.com/regebro/tzlocal	https://github.com/regebro/tzlocal/blob/master/LICENSE.txt
umf	https://github.com/oneapi-src/unified-memory-fram	https://github.com/oneapi-src/unified-memory-fram tab=License-1-ov-file#readme
umi-tools	https://github.com/CGATOxford/UMI-tools	https://github.com/CGATOxford/UMI-tools/blob/master/LICENSE
unicodedata2	https://github.com/fonttools/unicodedata2	https://github.com/fonttools/unicodedata2/blob/master/LICENSE
uri-template	https://gitlab.linss.com/open-source/python/uri-template	https://gitlab.linss.com/open-source/python/uri-template/-/blob/main/LICENSE
uritools	https://github.com/tkem/uritools/	https://github.com/tkem/uritools/blob/master/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL		
urllib3	https://urllib3.readthedocs.io/	https://github.com/urllib3/urllib3/blob/main/LICENSE.txt		
vine	https://github.com/celery/vine	https://github.com/celery/vine/blob/master/LICENSE		
vue	https://github.com/vuejs/vue	https://github.com/vuejs/vue/raw/master/LICENSE		
wadler_lindig	https://github.com/patrick-kidger/wadler_lindig	https://github.com/patrick-kidger/wadler_lindig?tab=Apache-2.0-1-ov-file#readme		
wadler_lindig	https://github.com/patrick-kidger/wadler_lindig	https://github.com/patrick-kidger/wadler_lindig?tab=Apache-2.0-1-ov-file#readmewcwidth	https://github.com/jquast/wcwidth	https://github.com/jquast/wcwidth/blob/master/LICENSE
wandb	https://github.com/wandb/wandb	https://github.com/wandb/wandb/blob/main/LICENSE		
webcolors	https://github.com/ubernostrum/webcolors	https://github.com/ubernostrum/webcolors?tab=BSD-3-Clause-1-ov-file#readme		
webencodings	https://github.com/gsnedders/python-webencodings	https://github.com/gsnedders/python-webencodings/blob/master/LICENSE		
websocket-client	https://github.com/websocket-client/websocket-client.git	https://github.com/websocket-client/websocket-client/blob/master/LICENSE		
Werkzeug	https://palletsprojects.com/p/werkzeug/	https://werkzeug.palletsprojects.com/en/2.1.x/license/		
westpa	http://github.com/westpa/westpa	https://github.com/westpa/westpa/blob/westpa2/LICENSE		
wheel	https://github.com/pypa/wheel	https://github.com/pypa/wheel/blob/main/LICENSE.txt		

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
widgetsnbextension	https://github.com/jupyter-widgets/ipywidgets#readme	https://github.com/jupyter-widgets/ipywidgets/blob/main/LICENSE
wrapt	https://github.com/GrahamDumpleton/wrapt	https://github.com/GrahamDumpleton/wrapt/blob/develop/LICENSE
wsutil	https://github.com/yhat/wsutil	https://github.com/yhat/wsutil/blob/master/LICENSE
x/lint	https://golang.org/x/lint	https://github.com/golang/lint/blob/master/LICENSE
x/mod	https://golang.org/x/mod/semver	https://cs.opensource.google/go/x/mod/+/master:LICENSE
x/net	https://golang.org/x/net	https://cs.opensource.google/go/x/net/+/master:LICENSE
x/oauth2	https://golang.org/x/oauth2	https://cs.opensource.google/go/x/oauth2/+/master:LICENSE
x/sys	https://golang.org/x/sys	https://cs.opensource.google/go/x/sys/+/master:LICENSE
x/text	https://golang.org/x/text	https://cs.opensource.google/go/x/text/+/master:LICENSE
x/tools	https://golang.org/x/tools	https://cs.opensource.google/go/x/tools/+/master:LICENSE
x/xerrors	https://golang.org/x/xerrors	https://cs.opensource.google/go/x/xerrors/+/master:LICENSE
xgboost	https://github.com/dmlc/xgboost	https://github.com/dmlc/xgboost/blob/master/LICENSE
xhtml2pdf	http://github.com/xhtml2pdf/xhtml2pdf	https://github.com/xhtml2pdf/xhtml2pdf/blob/master/LICENSE.txt

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
xlrd	https://github.com/python-excel/xlrd	https://github.com/python-excel/xlrd/blob/master/LICENSE
xlsxwriter	https://github.com/jmcnamara/XlsxWriter	https://github.com/jmcnamara/XlsxWriter/blob/main/LICENSE.txt
xmlsec	https://github.com/mehcode/python-xmlsec	https://github.com/mehcode/python-xmlsec/blob/master/LICENSE
xmltodict	https://github.com/martinblech/xmltodict	https://github.com/martinblech/xmltodict/blob/master/LICENSE
xorg-kbproto	https://gitlab.freedesktop.org/xorg/proto/kbproto	https://gitlab.freedesktop.org/xorg/proto/kbproto/-/blob/master/COPYING
xorg-libice	https://gitlab.freedesktop.org/xorg/lib/libice	https://gitlab.freedesktop.org/xorg/lib/libice/-/blob/master/COPYING
xorg-libsm	https://gitlab.freedesktop.org/xorg/lib/libsm	https://gitlab.freedesktop.org/xorg/lib/libsm/-/blob/master/COPYING
xorg-libx11	https://gitlab.freedesktop.org/xorg/lib/libx11	https://gitlab.freedesktop.org/xorg/lib/libx11/-/blob/master/COPYING
xorg-libxau	https://gitlab.freedesktop.org/xorg/lib/libxau	https://gitlab.freedesktop.org/xorg/lib/libxau/-/blob/master/COPYING
xorg-libxdmcp	https://gitlab.freedesktop.org/xorg/lib/libxdmcp	https://gitlab.freedesktop.org/xorg/lib/libxdmcp/-/blob/master/COPYING

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
xorg-libxext	https://gitlab.freedesktop.org/xorg/lib/libxext	https://gitlab.freedesktop.org/xorg/lib/libxext/-/blob/master/COPYING
xorg-libxrender	https://gitlab.freedesktop.org/xorg/lib/libxrender	https://gitlab.freedesktop.org/xorg/lib/libxrender/-/blob/master/COPYING
xorg-libxt	https://gitlab.freedesktop.org/xorg/lib/libxt	https://gitlab.freedesktop.org/xorg/lib/libxt/-/blob/master/COPYING
xorg-renderproto	https://gitlab.freedesktop.org/xorg/proto/renderproto	https://gitlab.freedesktop.org/xorg/proto/renderproto/-/blob/master/COPYING
xorg-xextproto	https://gitlab.freedesktop.org/xorg/proto/xextproto	https://gitlab.freedesktop.org/xorg/proto/xextproto/-/blob/master/COPYING
xorg-xproto	https://gitlab.freedesktop.org/xorg/proto/xproto	https://gitlab.freedesktop.org/xorg/proto/xproto/-/blob/master/COPYING
xxhash	https://github.com/cespare/xxhash/v2	https://github.com/cespare/xxhash/blob/main/LICENSE.txt
xz	https://github.com/ulikunitz/xz	https://github.com/ulikunitz/xz/blob/master/LICENSE
yaml	https://pyyaml.org/	https://github.com/yaml/pyyaml/blob/master/LICENSE
yaml-cpp	https://github.com/jbeder/yaml-cpp	https://github.com/jbeder/yaml-cpp/blob/master/LICENSE
yaml.v2	https://gopkg.in/yaml.v2	https://github.com/go-yaml/yaml/blob/v2.4.0/LICENSE
yaml.v3	https://gopkg.in/yaml.v3	https://github.com/go-yaml/yaml/blob/v3/LICENSE

continues on next page

Table 1 – continued from previous page

Name of Project	Website	License URL
yarl	https://github.com/aio-libs/yarl/	https://github.com/aio-libs/yarl/blob/master/LICENSE
yaspin	https://github.com/pavdmyt/yaspin	https://github.com/pavdmyt/yaspin/blob/master/LICENSE
yfiles	https://www.yworks.com/products/yfiles	commercial license
yml	https://pypi.org/project/yml/	N/A
zap	https://go.uber.org/zap	https://github.com/uber-go/zap/blob/master/LICENSE.txt
zipp	https://github.com/jaraco/zipp	https://github.com/jaraco/zipp/blob/main/LICENSE
zlib	https://zlib.net/	https://zlib.net/zlib_license.html
zstandard	https://github.com/indygreg/python-zstandard	https://github.com/indygreg/python-zstandard/blob/main/LICENSE
zstd	https://facebook.github.io/zstd/	https://github.com/facebook/zstd/blob/dev/LICENSE
click-option-group	https://github.com/click-contrib/click-option-group	https://github.com/click-contrib/click-option-group/blob/master/LICENSE

9.1 GCC

On the Linux platform, The OpenEye toolkits are linked with the GCC libraries using an eligible compilation process under the terms of the GCC RUNTIME LIBRARY EXCEPTION (see below).

The OpenEye toolkits are NOT open source software and are NOT governed by GPL or other open source licenses. Your right to use the toolkits are governed by the OpenEye license which is binding to You.

For certain Linux distributions, GCC library components are included as object files. Your rights to use GCC library components (libstdc++, libgcc, and libgomp) included with these distributions are governed by the GPLv3 license as follows:

The components libstdc++, libgcc, and libgomp are part of GCC.

- GCC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
- GCC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the

implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GCC (see below). If not, see Licenses GNU Project .

Per GPLv3, section 6d, the source for GCC is available for download from the following location: GNU. The GCC library components included here are from version gcc-4.9.3 and are built using the standard instructions at Installing GCC.

9.1.1 GCC RUNTIME LIBRARY EXCEPTION

```
Version 3.1, 31 March 2009

Copyright © 2009 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document,␣
↪but changing it is not allowed.

This GCC Runtime Library Exception ("Exception") is an additional permission under␣
↪section 7 of the GNU General Public License,
version 3 ("GPLv3"). It applies to a given file (the "Runtime Library") that bears a␣
↪notice placed by the copyright holder
of the file stating that the file is governed by GPLv3 along with this Exception.

When you use GCC to compile a program, GCC may combine portions of certain GCC header␣
↪files and runtime libraries with
the compiled program. The purpose of this Exception is to allow compilation of non-GPL␣
↪(including proprietary) programs
to use, in this way, the header files and runtime libraries covered by this Exception.

0. Definitions.

A file is an "Independent Module" if it either requires the Runtime Library for␣
↪execution after a Compilation Process,
or makes use of an interface provided by the Runtime Library, but is not otherwise based␣
↪on the Runtime Library.

"GCC" means a version of the GNU Compiler Collection, with or without modifications,␣
↪governed by version 3 (or a specified
later version) of the GNU General Public License (GPL) with the option of using any␣
↪subsequent versions published by the FSF.

"GPL-compatible Software" is software whose conditions of propagation, modification and␣
↪use would permit combination with
GCC in accord with the license of GCC.

"Target Code" refers to output from any compiler for a real or virtual target processor␣
↪architecture, in executable
form or suitable for input to an assembler, loader, linker and/or execution phase.␣
↪Notwithstanding that, Target Code
does not include data in any format that is used as a compiler intermediate␣
↪representation, or used for producing a
compiler intermediate representation.
```

(continues on next page)

(continued from previous page)

```
The "Compilation Process" transforms code entirely represented in non-intermediate␣
↪languages designed for human-written
code, and/or in Java Virtual Machine byte code, into Target Code. Thus, for example, use␣
↪of source code generators and
preprocessors need not be considered part of the Compilation Process, since the␣
↪Compilation Process can be understood
as starting with the output of the generators or preprocessors.

A Compilation Process is "Eligible" if it is done using GCC, alone or with other GPL-
↪compatible software, or if it is
done without using any work based on GCC. For example, using non-GPL-compatible Software␣
↪to optimize any GCC intermediate
representations would not qualify as an Eligible Compilation Process.

1. Grant of Additional Permission.

You have permission to propagate a work of Target Code formed by combining the Runtime␣
↪Library with Independent Modules,
even if such propagation would otherwise violate the terms of GPLv3, provided that all␣
↪Target Code was generated by
Eligible Compilation Processes. You may then convey such a combination under terms of␣
↪your choice, consistent with the
licensing of the Independent Modules.

2. No Weakening of GCC Copyleft.

The availability of this Exception does not imply any general presumption that third-
↪party software is unaffected by
the copyleft requirements of the license of GCC.
```

See also

- http://www.gnu.org/licenses/gcc-exception.html

9.1.2 GNU GENERAL PUBLIC LICENSE

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

 The GNU General Public License is a free, copyleft license for
 software and other kinds of works.

   The licenses for most software and other practical works are designed
```

(continues on next page)

(continued from previous page)

```
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
```

(continues on next page)

(continued from previous page)

```
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                    TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.
```

(continues on next page)

(continued from previous page)

```
  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
```

(continues on next page)

(continued from previous page)

```
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.


  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

   a) The work must carry prominent notices stating that you modified
   it, and giving a relevant date.

   b) The work must carry prominent notices stating that it is
   released under this License and any conditions added under section
   7.  This requirement modifies the requirement in section 4 to
   "keep intact all notices".
```

(continues on next page)

(continued from previous page)

```
   c) You must license the entire work, as a whole, under this
   License to anyone who comes into possession of a copy.  This
   License will therefore apply, along with any applicable section 7
   additional terms, to the whole of the work, and all its parts,
   regardless of how they are packaged.  This License gives no
   permission to license the work in any other way, but it does not
   invalidate such permission if you have separately received it.

   d) If the work has interactive user interfaces, each must display
   Appropriate Legal Notices; however, if the Program has interactive
   interfaces that do not display Appropriate Legal Notices, your
   work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

  a) Convey the object code in, or embodied in, a physical product
  (including a physical distribution medium), accompanied by the
  Corresponding Source fixed on a durable physical medium
  customarily used for software interchange.

  b) Convey the object code in, or embodied in, a physical product
  (including a physical distribution medium), accompanied by a
  written offer, valid for at least three years and valid for as
  long as you offer spare parts or customer support for that product
  model, to give anyone who possesses the object code either (1) a
  copy of the Corresponding Source for all the software in the
  product that is covered by this License, on a durable physical
  medium customarily used for software interchange, for a price no
  more than your reasonable cost of physically performing this
  conveying of source, or (2) access to copy the
  Corresponding Source from a network server at no charge.

  c) Convey individual copies of the object code with a copy of the
  written offer to provide the Corresponding Source.  This
  alternative is allowed only occasionally and noncommercially, and
  only if you received the object code with such an offer, in accord
```

(continues on next page)

(continued from previous page)

```
   with subsection 6b.

   d) Convey the object code by offering access from a designated
   place (gratis or for a charge), and offer equivalent access to the
   Corresponding Source in the same way through the same place at no
   further charge.  You need not require recipients to copy the
   Corresponding Source along with the object code.  If the place to
   copy the object code is a network server, the Corresponding Source
   may be on a different server (operated by you or a third party)
   that supports equivalent copying facilities, provided you maintain
   clear directions next to the object code saying where to find the
   Corresponding Source.  Regardless of what server hosts the
   Corresponding Source, you remain obligated to ensure that it is
   available for as long as needed to satisfy these requirements.

   e) Convey the object code using peer-to-peer transmission, provided
   you inform other peers where the object code and Corresponding
   Source of the work are being offered to the general public at no
   charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
```

(continues on next page)

(continued from previous page)

```
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
```

(continues on next page)

(continued from previous page)

```
   authors of the material; or

   e) Declining to grant rights under trademark law for use of some
   trade names, trademarks, or service marks; or

   f) Requiring indemnification of licensors and authors of that
   material by anyone who conveys the material (or modified versions of
   it) with contractual assumptions of liability to the recipient, for
   any liability that these contractual assumptions directly impose on
   those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.
```

(continues on next page)

(continued from previous page)

```
  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
```

(continues on next page)

```
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
```

(continues on next page)

(continued from previous page)

```
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
```

(continues on next page)

(continued from previous page)

```
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
```

(continues on next page)

(continued from previous page)

```
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

See also

- http://www.gnu.org/licenses/gpl.txt

INDEX

A

Aborted
    OESystem::OEConsoleProgressTracer, 1094
    OESystem::OETracerBase, 1175
AbsPath
    OEPlatform::OEFile, 1265
Acquire
    OEPlatform::OECondition, 1264
    OEPlatform::OEMutex, 1269
    OEPlatform::OESpinlock, 1271
    OEPlatform::OETryMutex, 1273
Add
    OEChem::OEAtomBondSet, 413
    OEChem::OEIsMember, 505
    OEChem::OEIsMemberPtr, 507
    OEChem::OEMolPropertyList, 585
    OEChem::OEVectorBindings, 657
AddAlias
    OESystem::OEParameter, 1152
AddAtom
    OEBio::OEInteractionHintFragment, 1373
    OEChem::OEAtomBondSet, 414
    OEChem::OEConfBase, 449
    OEChem::OEGroupBase, 471
AddAtoms
    OEChem::OEAtomBondSet, 414
AddAttribute
    OEChem::OECIFCategory, 431
AddBaseData
    OESystem::OEBase, 1063
AddBond
    OEChem::OEAtomBase, 399
    OEChem::OEAtomBondSet, 414
    OEChem::OEConfBase, 449
    OEChem::OEGroupBase, 472
AddBonds
    OEChem::OEAtomBondSet, 414
AddCategory
    OEChem::OECIFData, 435
AddChild
    OESystem::OEAnnotation, 1060
AddCode
    OEChem::OEMonomerData, 1030
AddCodeSet
    OEChem::OEMonomerSet, 1034
AddColor
    OEChem::OEMolStyleColorer, 590
AddConstraint
    OEChem::OECliqueSearch, 445
    OEChem::OEMCSSearch, 546
    OEChem::OESubSearch, 626
AddData
    OEChem::OECIFData, 436
    OESystem::OEBase, 1063
AddDefault
    OESystem::OETypedParameter, 1177
AddDetail
    OESystem::OEInterface, 1134
    OESystem::OEOptions, 1253
    OESystem::OEParameter, 1152
AddGaussian
    OESystem::OEGaussianContainer, 1114
    OESystem::OEGaussianContainerBase, 1116
AddHistory
    OESystem::OEHeader, 1131
AddIllegalRange
    OESystem::OEParameter, 1152
AddIllegalValue
    OESystem::OEParameter, 1152
AddInterface
    OESystem::OEInterface, 1134
AddLegalEntry
    OESystem::OEConstParameter, 1246
    OESystem::OEMultiParameter, 1251
AddLegalRange
    OESystem::OEParameter, 1152
AddLegalValue
    OESystem::OEParameter, 1153
AddLigandName
    OEBio::OEMolComplexCategorizer, 1414
AddMolecule
    OEBio::OEInteractionHintContainer, 1369
AddMonomer
    OEChem::OEMonomerSet, 1034

AddNewFeature
    OEBio::OEReceptorCustomConstraints, 1458
AddNewSphere
    OEBio::OEReceptorConstraintFeature, 1456
AddOption
    OESystem::OEOptions, 1253
AddPair
    OEChem::OEMatch, 529
AddParameter
    OESystem::OEInterface, 1134
    OESystem::OEOptions, 1253
AddParameterPrefix
    OESystem::OEOptions, 1253
AddParameterSuffix
    OESystem::OEOptions, 1253
AddProteinConstraint
    OEBio::OEReceptor, 1452
AddRings
    OEChem::OE2DRingDictionary, 382
AddRole
    OESystem::OERoleSet, 1162
AddRow
    OEChem::OECIFCategory, 431
    OEChem::OECIFData, 435
    OEMath::OEMatrix, 1305
AddSmarts
    OEBio::OEReceptorConstraintFeature, 1456
AddStartingMaterial
    OEChem::OELibraryGen, 520
AddStop
    OESystem::OEColorGradientBase, 1090
addstream
    OEPlatform::oemultiostream, 1283
AddStringDefault
    OESystem::OEParameter, 1153
AddStringValue
    OESystem::OEParameter, 1153
AddToDB
    OEBio::OEResidueCategoryData, 1460
AddValue
    OESystem::OETypedParameter, 1177
AIER, 227
Alignment, 216
AllInDB
    OEBio::OEResidueCategoryData, 1461
Alternate Locations, 225
AltLoc AltConf OEAltLocationFactory, 218
anisou.cs
    Example Code, 343
append
    OEPlatform::oeofstream, 1284
appendringdict.cs
    Example Code, 338
aromatic ring system identification, 56
AsRegularGrid
    OESystem::OESkewGrid, 1168
atom aromaticity, 44
atom chirality, 44
atom connection, 44
atom degree, 44
atom heavy degree, 44
atom heavy valence, 44
atom name, 44
atom parent, 44
atom properties, 43
atom radius, 44
atom ring membership, 44
atom stereo, 44
atom type name, 44
atom valence, 44
atomic hybridization, 44
atomic number, 44, 176
atomic symbol, 176
AtomMatch
    OEChem::OESubSearch, 627
AutoMapRxn
    OEChem::OERxnAutomapper, 1039

B

backbone.cs
    Example Code, 350
Beta Sheet, 216
Biopolymer Residue, 211
BLOSUM62, 216
bond aromaticity, 46
bond begin atom, 46
bond begin atom index, 46
bond chirality, 46
bond end atom, 46
bond end atom index, 46
bond index, 46
bond order, 46
bond parent, 46
bond properties, 46
bond ring membership, 46
bond stereo, 46
bond type name, 46
Broadcast
    OEPlatform::OECondition, 1264
BuildExpressions
    OEChem::OEQMol, 608
    OEChem::OEQMolBase, 610

C

CalculateRMSD
    OEChem::OEConfRMSD, 453
Cancel
    OEChem::OESubSearchResult, 636

cansmi.cs
    Example Code, 255
catmols.cs
    Example Code, 244
ChemCompToMol
    OEChem::OECIFData, 436
CIF data, 30
Classification, 218
Clear
    OEBio::OEDesignUnit, 1346
    OEBio::OEIridiumData, 1377
    OEBio::OEReceptor, 1452
    OEBio::OEReceptorCustomConstraints, 1458
    OEBio::OESequenceAlignment, 1464
    OEBio::OEStructureQuality, 1474
    OEChem::OEAtomBase, 399
    OEChem::OEAtomBondSet, 414
    OEChem::OEBondBase, 422
    OEChem::OECSVHeader, 461
    OEChem::OEGroupBase, 472
    OEChem::OEMatch, 530
    OEChem::OEMatchBase, 531
    OEChem::OEMolBase, 553
    OEChem::OEMolDatabase, 564
    OEChem::OEMolPropertyList, 585
    OEChem::OETrans, 642
    OEMath::OEMatrix, 1305
    OESystem::OEBase, 1063
    OESystem::OEBinaryIOHandler, 1072
    OESystem::OEBinaryIOHandlerBase, 1074
    OESystem::OEErrorHandler, 1098
    OESystem::OEErrorHandlerImplBase, 1101
    OESystem::OEGaussianContainer, 1114
    OESystem::OEGaussianContainerBase, 1116
    OESystem::OEGrid, 1119
    OESystem::OEGridAbstractBase, 1121
    OESystem::OEGridBase, 1123
    OESystem::OEInterface, 1134
    OESystem::OEMultiGrid, 1146
    OESystem::OEOptions, 1253
    OESystem::OEScalarGrid, 1165
    OESystem::OESkewGrid, 1169
clear
    OEPlatform::oeisstream, 1277
    OEPlatform::oeistdstream, 1278
    OEPlatform::oeiwrapperstream, 1282
    OEPlatform::oemultiostream, 1283
    OEPlatform::oeosstream, 1286
    OEPlatform::oeostdstream, 1286
    OEPlatform::oeowrapperstream, 1289
ClearAtomColorer
    OEChem::OE3DMolStyle, 385
ClearAtomLabelColorer
    OEChem::OE3DMolStyle, 386
ClearAtomLabelDataTag
    OEChem::OE3DMolStyle, 386
ClearAtomLabelText
    OEChem::OE3DMolStyle, 386
ClearAtomLabelType
    OEChem::OE3DMolStyle, 386
ClearAtoms
    OEChem::OEAtomBondSet, 415
ClearAtomsAndBonds
    OEChem::OEAtomBondSet, 415
ClearAtomStyle
    OEChem::OE3DMolStyle, 386
ClearBase
    OEChem::OEMCMolBase, 535
    OEChem::OEMol, 552
    OEChem::OEMolBase, 553
ClearBits
    OESystem::OEBitVector, 1078
ClearBonds
    OEChem::OEAtomBondSet, 415
ClearColors
    OEChem::OEMolStyleColorer, 590
ClearComponent
    OEBio::OEDesignUnit, 1348
ClearConstraints
    OEChem::OECliqueSearch, 445
    OEChem::OEMCSSearch, 547
    OEChem::OESubSearch, 627
ClearContourStyle
    OEChem::OE3DMolStyle, 386
ClearCoords
    OEChem::OEConfBase, 449
    OEChem::OEMolBase, 553
ClearCSVHeader
    OEChem::oemolstreambase, 587
ClearCustomConstraints
    OEBio::OEReceptor, 1452
ClearDBCategory
    OEBio::OEResidueCategoryData, 1460
ClearDefaults
    OESystem::OEParameter, 1153
ClearGridContourColorer
    OEChem::OE3DMolStyle, 386
ClearGridType
    OEChem::OE3DMolStyle, 386
ClearHydrogenVisibility
    OEChem::OE3DMolStyle, 386
ClearInteractionStyle
    OEChem::OE3DMolStyle, 387
ClearLigandSubSearch
    OEBio::OEMolComplexCategorizer, 1415
ClearMCMol
    OEChem::OEMCMolBase, 535
    OEChem::OEMol, 552

ClearMolBase
    OEChem::OEMCMolBase, 535
    OEChem::OEMol, 552
ClearProteinColorer
    OEChem::OE3DMolStyle, 387
ClearProteinConstraint
    OEBio::OEReceptor, 1452
ClearProteinConstraints
    OEBio::OEReceptor, 1452
ClearProteinStyle
    OEChem::OE3DMolStyle, 387
ClearReceptor
    OEBio::OEDesignUnit, 1348
ClearRoles
    OESystem::OERoleSet, 1162
ClearSmarts
    OEBio::OEReceptorConstraintFeature, 1457
ClearSpheres
    OEBio::OEReceptorConstraintFeature, 1456
ClearStartingMaterial
    OEChem::OELibraryGen, 521
ClearSurfaceAlpha
    OEChem::OE3DMolStyle, 387
ClearSurfaceColorer
    OEChem::OE3DMolStyle, 387
ClearSurfaceStyle
    OEChem::OE3DMolStyle, 387
ClearSurfaceType
    OEChem::OE3DMolStyle, 387
ClearValues
    OESystem::OEParameter, 1153
cliquealign.cs
    Example Code, 294
Close
    OEPlatform::OEDirectoryScan, 1265
close
    OEChem::oemolistream, 574
    OEChem::oemolithread, 577
    OEChem::oemolostream, 579
    OEChem::oemolothread, 580
    OEChem::oemolstreambase, 588
    OEPlatform::oeistream, 1279
    OEPlatform::oeostream, 1287
ClusterHasBadClash
    OEBio::OEPlaceHydrogensDetails, 1434
CombineMols
    OEChem::OEAbsCanonicalConfTest, 396
    OEChem::OEAbsoluteConfTest, 397
    OEChem::OEConfTestBase, 457
    OEChem::OEDefaultConfTest, 462
    OEChem::OEIsomericConfTest, 509
    OEChem::OEOmegaConfTest, 597
command line interface definition, 196
command line options parsing, 190
CompareMols
    OEChem::OEAbsCanonicalConfTest, 396
    OEChem::OEAbsoluteConfTest, 397
    OEChem::OEConfTestBase, 457
    OEChem::OEDefaultConfTest, 462
    OEChem::OEIsomericConfTest, 510
    OEChem::OEOmegaConfTest, 597
Compress
    OEChem::OEMolBase, 553
    OEChem::OETrans, 642
connected components identification, 56
Constructor
    OEBio::OEReceptor, 1452
    OEBio::OEReceptorConstraintFeature, 1455
    OEMath::OEKernelPLS, 1304
    OEMath::OESphere, 1306
Constructors
    OEBio::OEAltGroup, 1325
    OEBio::OEAltLocation, 1326
    OEBio::OEAltLocationFactory, 1327
    OEBio::OEAltLocationState, 1333
    OEBio::OEAtomMatchResidue, 1334
    OEBio::OEAtomMatchResidueID, 1335
    OEBio::OECationPiInteractionHint, 1337
    OEBio::OEChelatorInteractionHint, 1339
    OEBio::OEClashInteractionHint, 1340
    OEBio::OEContactInteractionHint, 1341
    OEBio::OECovalentInteractionHint, 1342
    OEBio::OECrystalSymmetryParams, 1343
    OEBio::OEDesignUnit, 1345
    OEBio::OEHalogenBondInteractionHint, 1348
    OEBio::OEHasAlternateLocation, 1350
    OEBio::OEHasInteractionHint, 1350
    OEBio::OEHasInteractionHintType, 1351
    OEBio::OEHasPDBAtomIndex, 1352
    OEBio::OEHasResidueInteractionHint, 1353
    OEBio::OEHasSecondaryStructure, 1354
    OEBio::OEHBondInteractionHint, 1355
    OEBio::OEHierChain, 1356
    OEBio::OEHierFragment, 1357
    OEBio::OEHierResidue, 1358
    OEBio::OEHierView, 1359
    OEBio::OEInteractionHint, 1361
    OEBio::OEInteractionHintComponentTypeBase, 1367
    OEBio::OEInteractionHintContainer, 1368
    OEBio::OEInteractionHintFragment, 1373
    OEBio::OEInteractionHintTypeBase, 1375
    OEBio::OEIridiumData, 1376
    OEBio::OEIsBackboneAtom, 1381
    OEBio::OEIsCationPiInteractionHint, 1382
    OEBio::OEIsChelatorInteractionHint, 1383
    OEBio::OEIsClashingHBondInteractionHint, 1384

OEBio::OEIsClashInteractionHint, 1384
OEBio::OEIsContactInteractionHint, 1385
OEBio::OEIsCovalentInteractionHint, 1386
OEBio::OEIsCTerminalAtom, 1386
OEBio::OEIsDisulfideBond, 1387
OEBio::OEIsHalogenBondInteractionHint, 1388
OEBio::OEIsHBondInteractionHint, 1389
OEBio::OEIsHetAtom, 1390
OEBio::OEIsInterInteractionHint, 1390
OEBio::OEIsIntermolecularCationPiInteractionHint, 1391
OEBio::OEIsIntermolecularChelatorInteractionHint, 1392
OEBio::OEIsIntermolecularHalogenBondInteractionHint, 1393
OEBio::OEIsIntermolecularHBondInteractionHint, 1394
OEBio::OEIsIntermolecularSaltBridgeInteractionHint, 1395
OEBio::OEIsIntermolecularStackingInteractionHint, 1396
OEBio::OEIsIntraInteractionHint, 1396
OEBio::OEIsIntramolecularCationPiInteractionHint, 1397
OEBio::OEIsIntramolecularChelatorInteractionHint, 1398
OEBio::OEIsIntramolecularHalogenBondInteractionHint, 1399
OEBio::OEIsIntramolecularHBondInteractionHint, 1400
OEBio::OEIsIntramolecularSaltBridgeInteractionHint, 1401
OEBio::OEIsIntramolecularStackingInteractionHint, 1401
OEBio::OEIsNonIdealHBondInteractionHint, 1402
OEBio::OEIsNTerminalAtom, 1403
OEBio::OEIsSaltBridgeInteractionHint, 1406
OEBio::OEIsStackingInteractionHint, 1407
OEBio::OEIsStandardAminoAcid, 1408
OEBio::OEIsStandardNucleicAcid, 1409
OEBio::OEIsUnpairedInteractionHint, 1410
OEBio::OEIsUnpairedLigandInteractionHint, 1410
OEBio::OEIsUnpairedProteinInteractionHint, 1411
OEBio::OEIsWater, 1412
OEBio::OELigandInteractionHintComponent, 1413
OEBio::OEMolComplexCategorizer, 1414
OEBio::OEMolComplexFilter, 1419
OEBio::OEPerceiveInteractionOptions, 1421
OEBio::OEPlaceHydrogensDetails, 1434
OEBio::OEPlaceHydrogensMoverNote, 1438
OEBio::OEPlaceHydrogensOptions, 1442
OEBio::OEProteinInteractionHintComponent, 1448
OEBio::OERamachandranAnalysis, 1450
OEBio::OEReceptorCustomConstraints, 1458
OEBio::OEResidueCategoryData, 1460
OEBio::OESaltBridgeInteractionHint, 1463
OEBio::OESequenceAlignment, 1464
OEBio::OESplitMolComplexOptions, 1466
OEBio::OEStackingInteractionHint, 1472
OEBio::OEStructureQuality, 1473
OEChem::OE2DRingDictionary, 381
OEChem::OE2DRingDictionaryCreatorOptions, 382
OEChem::OE3DMolStyle, 385
OEChem::OEAbsoluteConfTest, 397
OEChem::OEAnisoUij, 398
OEChem::OEAtomBondSet, 413
OEChem::OEAtomIsInResidue, 420
OEChem::OECIFCategory, 431
OEChem::OECIFData, 435
OEChem::OECIFOptions, 442
OEChem::OECliqueSearch, 444
OEChem::OEConfRMSD, 453
OEChem::OEConfRMSDOptions, 456
OEChem::OEConstCoords, 458
OEChem::OECoordArray, 458
OEChem::OECreateSubSearchDatabaseOptions, 459
OEChem::OECSVHeader, 461
OEChem::OEEuler, 463
OEChem::OEExprBase, 464
OEChem::OEExtractRingTemplateOptions, 466
OEChem::OEFileStringParameter, 468
OEChem::OEFuzzy, 469
OEChem::OEGetNbrAtom, 470
OEChem::OEGraphMol, 471
OEChem::OEHasAlphaBetaUnsat, 475
OEChem::OEHasAminoAcidType, 1019
OEChem::OEHasAtomicNum, 476
OEChem::OEHasAtomInGroup, 476
OEChem::OEHasAtomName, 477
OEChem::OEHasAtomStereoSpecified, 478
OEChem::OEHasBondInGroup, 479
OEChem::OEHasBondStereoSpecified, 480
OEChem::OEHasChainID, 480
OEChem::OEHasFormalCharge, 481
OEChem::OEHasFragmentNumber, 482
OEChem::OEHasGroupType, 483
OEChem::OEHasHvyDegree, 483
OEChem::OEHasIdx, 484
OEChem::OEHasMapIdx, 485

OEChem::OEHasMonomerGroup, 1020
OEChem::OEHasMonomerType, 1020
OEChem::OEHasOrder, 486
OEChem::OEHasPolymerType, 1021
OEChem::OEHasResidueNumber, 486
OEChem::OEHelmGenerationOptions, 1021
OEChem::OEHelmGenerationResult, 1023
OEChem::OEHelmParsingResult, 1024
OEChem::OEHydrogenCounter, 487
OEChem::OEIdxSelected, 488
OEChem::OEInChIOptions, 489
OEChem::OEInSamePart, 492
OEChem::OEIsAmideBond, 493
OEChem::OEIsAtomHybridization, 495
OEChem::OEIsDisulfideBridgeBond, 499
OEChem::OEIsInMonomerCodeSet, 1025
OEChem::OEIsIntraResidueBond, 502
OEChem::OEIsMDLStereoGroup, 504
OEChem::OEIsMember, 505
OEChem::OEIsMemberPtr, 506
OEChem::OEIsMonomerGroup, 1026
OEChem::OEIsomericConfTest, 509
OEChem::OEIsProductGroup, 512
OEChem::OEIsReactantGroup, 513
OEChem::OEIsRGroup, 514
OEChem::OEIsRotor, 515
OEChem::OEIsRxnComponentGroup, 516
OEChem::OELibraryGen, 519
OEChem::OELingoSim, 527
OEChem::OEMatch, 529
OEChem::OEMatchFunc, 533
OEChem::OEMatchPair, 534
OEChem::OEMCSFuncParameter, 541
OEChem::OEMCSMaxAtomsCompleteCycles, 543
OEChem::OEMCSMaxBondsCompleteCycles, 545
OEChem::OEMCSSearch, 546
OEChem::OEMMFFTypeResult, 550
OEChem::OEMol, 551
OEChem::OEMolDatabase, 563
OEChem::OEMolDatabaseSaveOptions, 568
OEChem::OEMolDBException, 570
OEChem::OEMolGDPropertyFxn, 570
OEChem::oemolistream, 571
OEChem::oemolithread, 576
OEChem::oemolostream, 577
OEChem::oemolothread, 580
OEChem::OEMolParameter, 581
OEChem::OEMolPropertyList, 585
OEChem::OEMolReaderParameter, 583
OEChem::OEMolSDPropertyFxn, 586
OEChem::OEMolStyleColorer, 589
OEChem::OEMolTaggedPropertyFxn, 592
OEChem::OEMolWriterParameter, 584
OEChem::OEMonomer, 1026
OEChem::OEMonomerData, 1029
OEChem::OEMonomerSet, 1034
OEChem::OEMonomerValidationResult, 1038
OEChem::OEMutableCoords, 594
OEChem::OENbrs, 594
OEChem::OENearestNbrs, 595
OEChem::OENthAtom, 596
OEChem::OEOmegaConfTest, 597
OEChem::OEOutFileStringParameter, 598
OEChem::OEParseSmilesOptions, 599
OEChem::OEPartPredT, 601
OEChem::OEPredicateParameter, 602
OEChem::OEQMol, 607
OEChem::OEQMolBase, 610
OEChem::OEQuaternion, 612
OEChem::OERefInputAppOptions, 613
OEChem::OEResidue, 615
OEChem::OERotMatrix, 621
OEChem::OERxnAutomapper, 1038
OEChem::OERxnAutomapperOptions, 1039
OEChem::OESDDataPair, 621
OEChem::OESimilarByRMSD, 622
OEChem::OESimpleAppOptions, 623
OEChem::OESubSearch, 625
OEChem::OESubSearchDatabase, 631
OEChem::OESubSearchQuery, 634
OEChem::OESubSearchResult, 636
OEChem::OESubSearchScreen, 638
OEChem::OETorsion, 641
OEChem::OETrans, 641
OEChem::OETranslation, 644
OEChem::OEUnaryToBinaryAnd, 645
OEChem::OEUnaryToBinaryOr, 646
OEChem::OEUnAtomPredParameter, 647
OEChem::OEUnBondPredParameter, 648
OEChem::OEUniMolecularRxn, 649
OEChem::OEUniMolecularRxnOptions, 653
OEChem::OEVectorBindings, 657
OEMath::OEBox, 1301
OEMath::OEMatrix, 1305
OEMath::OESquareMatrix, 1308
OEPlatform::OECondition, 1264
OEPlatform::OEDirectoryScan, 1264
OEPlatform::OEFile, 1265
OEPlatform::OEHalfFloat, 1267
OEPlatform::oeifstream, 1274
OEPlatform::oeigzstream, 1275
OEPlatform::oeiofstream, 1276
OEPlatform::oeisstream, 1277
OEPlatform::oeistdstream, 1278
OEPlatform::oeistream, 1279
OEPlatform::oeiwrapperstream, 1282
OEPlatform::OELockCondition, 1268
OEPlatform::oemultiostream, 1283

OEPlatform::OEMutex, 1269
OEPlatform::oeofstream, 1284
OEPlatform::oeogzstream, 1285
OEPlatform::oeosstream, 1285
OEPlatform::oeostdstream, 1286
OEPlatform::oeostream, 1287
OEPlatform::oeowrapperstream, 1289
OEPlatform::OESpinlock, 1271
OEPlatform::oestream, 1290
OEPlatform::OEThread, 1271
OEPlatform::OEThreadLocal, 1273
OEPlatform::OETryMutex, 1273
OESystem::OEAnd, 1059
OESystem::OEAnnotation, 1060
OESystem::OEBase, 1062
OESystem::OEBaseData, 1067
OESystem::OEBaseDataT, 1069
OESystem::OEBinaryAnd, 1070
OESystem::OEBinaryIOHandler, 1072
OESystem::OEBinaryNot, 1075
OESystem::OEBinaryOr, 1076
OESystem::OEBInitializer, 1061
OESystem::OEBitVector, 1078
OESystem::OEBoolParameter, 1245
OESystem::OEBoundedBuffer, 1082
OESystem::OEColor, 1085
OESystem::OEColorGradientBase, 1090
OESystem::OEColorStop, 1092
OESystem::OEConcatIter, 1093
OESystem::OEConsoleProgressTracer, 1093
OESystem::OEConstParameter, 1246
OESystem::OEContainer, 1095
OESystem::OECycleTimer, 1095
OESystem::OEDots, 1096
OESystem::OEDoubleParameter, 1247
OESystem::OEErrorHandler, 1098
OESystem::OEErrorHandlerImplBase, 1101
OESystem::OEExponentColorGradient, 1103
OESystem::OEExponentialColorGradient, 1105
OESystem::OEFloatParameter, 1248
OESystem::OEGaussian, 1110
OESystem::OEGaussianContainer, 1114
OESystem::OEGrid, 1118
OESystem::OEHasRole, 1130
OESystem::OEHeader, 1131
OESystem::OEInterface, 1133
OESystem::OEIntParameter, 1250
OESystem::OEIter, 1139
OESystem::OEIterBase, 1141
OESystem::OELinearColorGradient, 1142
OESystem::OELogarithmicColorGradient, 1144
OESystem::OEMultiGrid, 1145
OESystem::OEMultiParameter, 1251
OESystem::OENot, 1148
OESystem::OEOr, 1149
OESystem::OEOwnedPtr, 1150
OESystem::OEPredIter, 1158
OESystem::oeprogressstream, 1186
OESystem::OEProtectedBuffer, 1159
OESystem::OERandom, 1160
OESystem::OERole, 1161
OESystem::OERoles, 1163
OESystem::OEScalarGrid, 1165
OESystem::OESharedPtr, 1167
OESystem::OESkewGrid, 1168
OESystem::OEStopwatch, 1173
OESystem::OEStringParameter, 1259
OESystem::OEThreadedDots, 1174
OESystem::OEUIntParameter, 1261
OESystem::OEUnboundedBuffer, 1184
OESystem::OEUnownedPtr, 1185
OESystem::OEWallTimer, 1185
convert.cs
Example Code, 243
Copy
OESystem::OEConcatIter, 1093
OESystem::OEIter, 1140
OESystem::OEIterBase, 1141
OESystem::OEPredIter, 1158
Count
OEChem::OEMolBase, 553
OESystem::OEErrorHandler, 1098
OESystem::OEErrorHandlerImplBase, 1102
CountBits
OESystem::OEBitVector, 1078
CountRangeBits
OESystem::OEBitVector, 1078
CreateCopy
OEBio::OEAtomMatchResidue, 1334
OEBio::OECationPiInteractionHint, 1338
OEBio::OEChelatorInteractionHint, 1339
OEBio::OEClashInteractionHint, 1340
OEBio::OEContactInteractionHint, 1341
OEBio::OECovalentInteractionHint, 1342
OEBio::OEDesignUnit, 1346
OEBio::OEHalogenBondInteractionHint, 1349
OEBio::OEHasAlternateLocation, 1350
OEBio::OEHasInteractionHint, 1351
OEBio::OEHasInteractionHintType, 1352
OEBio::OEHasPDBAtomIndex, 1352
OEBio::OEHasResidueInteractionHint, 1353
OEBio::OEHasSecondaryStructure, 1354
OEBio::OEHBondInteractionHint, 1355
OEBio::OEInteractionHint, 1362
OEBio::OEInteractionHintComponentTypeBase, 1367

OEBio::OEInteractionHintFragment, 1373
OEBio::OEInteractionHintTypeBase, 1375
OEBio::OEIsBackboneAtom, 1382
OEBio::OEIsCationPiInteractionHint, 1382
OEBio::OEIsChelatorInteractionHint, 1383
OEBio::OEIsClashingHBondInteractionHint, 1384
OEBio::OEIsClashInteractionHint, 1385
OEBio::OEIsContactInteractionHint, 1385
OEBio::OEIsCovalentInteractionHint, 1386
OEBio::OEIsCTerminalAtom, 1387
OEBio::OEIsDisulfideBond, 1387
OEBio::OEIsHalogenBondInteractionHint, 1388
OEBio::OEIsHBondInteractionHint, 1389
OEBio::OEIsHetAtom, 1390
OEBio::OEIsInterInteractionHint, 1391
OEBio::OEIsIntermolecularCationPiInteractionHint, 1391
OEBio::OEIsIntermolecularChelatorInteractionHint, 1392
OEBio::OEIsIntermolecularHalogenBondInteractionHint, 1393
OEBio::OEIsIntermolecularHBondInteractionHint, 1394
OEBio::OEIsIntermolecularSaltBridgeInteractionHint, 1395
OEBio::OEIsIntermolecularStackingInteractionHint, 1396
OEBio::OEIsIntraInteractionHint, 1397
OEBio::OEIsIntramolecularCationPiInteractionHint, 1397
OEBio::OEIsIntramolecularChelatorInteractionHint, 1398
OEBio::OEIsIntramolecularHalogenBondInteractionHint, 1399
OEBio::OEIsIntramolecularHBondInteractionHint, 1400
OEBio::OEIsIntramolecularSaltBridgeInteractionHint, 1401
OEBio::OEIsIntramolecularStackingInteractionHint, 1402
OEBio::OEIsNonIdealHBondInteractionHint, 1403
OEBio::OEIsNTerminalAtom, 1403
OEBio::OEIsNucleicAcidBase, 1404
OEBio::OEIsNucleicAcidPhosphate, 1405
OEBio::OEIsNucleicAcidSugar, 1406
OEBio::OEIsSaltBridgeInteractionHint, 1407
OEBio::OEIsStackingInteractionHint, 1408
OEBio::OEIsStandardAminoAcid, 1408
OEBio::OEIsStandardNucleicAcid, 1409
OEBio::OEIsUnpairedInteractionHint, 1410
OEBio::OEIsUnpairedLigandInteractionHint, 1411
OEBio::OEIsUnpairedProteinInteractionHint, 1411
OEBio::OEIsWater, 1412
OEBio::OELigandInteractionHintComponent, 1413
OEBio::OEMolComplexCategorizer, 1415
OEBio::OEMolComplexCategorizerBase, 1419
OEBio::OEMolComplexFilter, 1420
OEBio::OEProteinInteractionHintComponent, 1448
OEBio::OEReceptorConstraintFeature, 1457
OEBio::OESaltBridgeInteractionHint, 1463
OEBio::OEStackingInteractionHint, 1473
OEChem::OEAbsCanonicalConfTest, 396
OEChem::OEAbsoluteConfTest, 397
OEChem::OEAtomBondSet, 415
OEChem::OEAtomIsInAgent, 418
OEChem::OEAtomIsInChain, 419
OEChem::OEAtomIsInProduct, 419
OEChem::OEAtomIsInReactant, 420
OEChem::OEAtomIsInResidue, 421
OEChem::OEAtomIsInRing, 421
OEChem::OEBondIsInChain, 430
OEChem::OEBondIsInRing, 430
OEChem::OEConfTestBase, 457
OEChem::OEDefaultConfTest, 463
OEChem::OEEuler, 463
OEChem::OEExprBase, 465
OEChem::OEGetNbrAtom, 470
OEChem::OEHasAlphaBetaUnsat, 475
OEChem::OEHasAminoAcidType, 1019
OEChem::OEHasAnisou, 475
OEChem::OEHasAtomicNum, 476
OEChem::OEHasAtomInGroup, 477
OEChem::OEHasAtomName, 477
OEChem::OEHasAtomStereoSpecified, 478
OEChem::OEHasBondInGroup, 479
OEChem::OEHasBondStereoSpecified, 480
OEChem::OEHasChainID, 481
OEChem::OEHasFormalCharge, 481
OEChem::OEHasFragmentNumber, 482
OEChem::OEHasGroupType, 483
OEChem::OEHasHvyDegree, 483
OEChem::OEHasIdx, 484
OEChem::OEHasMapIdx, 485
OEChem::OEHasMonomerGroup, 1020
OEChem::OEHasMonomerType, 1020
OEChem::OEHasOrder, 486
OEChem::OEHasPolymerType, 1021
OEChem::OEHasResidueNumber, 487
OEChem::OEIdxSelected, 488
OEChem::OEInSamePart, 492

OEChem::OEIsAmideBond, 493
OEChem::OEIsAromaticAtom, 494
OEChem::OEIsAromaticBond, 494
OEChem::OEIsAtomHybridization, 495
OEChem::OEIsBridgeHead, 496
OEChem::OEIsCAlpha, 496
OEChem::OEIsCarbon, 497
OEChem::OEIsChiralAtom, 498
OEChem::OEIsChiralBond, 498
OEChem::OEIsDisulfideBridgeBond, 499
OEChem::OEIsHalide, 499
OEChem::OEIsHalogen, 500
OEChem::OEIsHeavy, 501
OEChem::OEIsHetero, 501
OEChem::OEIsHydrogen, 502
OEChem::OEIsInMonomerCodeSet, 1025
OEChem::OEIsIntraResidueBond, 503
OEChem::OEIsInvertibleNitrogen, 503
OEChem::OEIsMDLStereoGroup, 504
OEChem::OEIsMember, 505
OEChem::OEIsMemberPtr, 507
OEChem::OEIsMetal, 507
OEChem::OEIsMonomerGroup, 1026
OEChem::OEIsNitrogen, 508
OEChem::OEIsNonRingAtomDoubleBondedToRing, 508
OEChem::OEIsomericConfTest, 510
OEChem::OEIsOxygen, 510
OEChem::OEIsPhosphorus, 511
OEChem::OEIsPolar, 511
OEChem::OEIsPolarHydrogen, 512
OEChem::OEIsProductGroup, 513
OEChem::OEIsReactantGroup, 514
OEChem::OEIsRGroup, 514
OEChem::OEIsRotor, 515
OEChem::OEIsRxnComponentGroup, 516
OEChem::OEIsSulfur, 517
OEChem::OEIsTermHeavyAtom, 517
OEChem::OEIsValidAtomValence, 518
OEChem::OEIsValidMDLAtomValence, 519
OEChem::OEMatch, 530
OEChem::OEMatchBase, 531
OEChem::OEMatchFunc, 534
OEChem::OEMCSFunc, 540
OEChem::OEMCSMaxAtoms, 542
OEChem::OEMCSMaxAtomsCompleteCycles, 543
OEChem::OEMCSMaxBonds, 544
OEChem::OEMCSMaxBondsCompleteCycles, 545
OEChem::OEMolBase, 554
OEChem::OEMolDatabase, 564
OEChem::OEMolPropertyFxn, 584
OEChem::OEMolPropertyList, 585
OEChem::OENthAtom, 596
OEChem::OEOmegaConfTest, 597
OEChem::OEPartPredT, 601
OEChem::OEQMolBase, 611
OEChem::OEQuaternion, 612
OEChem::OERotMatrix, 621
OEChem::OESimilarByRMSD, 623
OEChem::OETransBase, 643
OEChem::OETranslation, 644
OEChem::OEUnaryToBinaryAnd, 645
OEChem::OEUnaryToBinaryOr, 646
OESystem::OEAnd, 1059
OESystem::OEAnnotation, 1060
OESystem::OEBase, 1063
OESystem::OEBaseData, 1068
OESystem::OEBaseDataT, 1069
OESystem::OEBinaryAnd, 1071
OESystem::OEBinaryFalse, 1071
OESystem::OEBinaryFunction, 1072
OESystem::OEBinaryIOHandler, 1072
OESystem::OEBinaryNot, 1076
OESystem::OEBinaryOr, 1076
OESystem::OEBinaryPredicate, 1077
OESystem::OEBinaryTrue, 1077
OESystem::OEColor, 1085
OESystem::OEContainer, 1095
OESystem::OEErrorHandlerImplBase, 1102
OESystem::OEFixedGrid, 1107
OESystem::OEFixedMultiGrid, 1109
OESystem::OEGaussian, 1110
OESystem::OEGaussianContainer, 1115
OESystem::OEGrid, 1119
OESystem::OEGridBase, 1123
OESystem::OEHasRole, 1130
OESystem::OEIsFalse, 1138
OESystem::OEIsTrue, 1139
OESystem::OEMultiGrid, 1146
OESystem::OENot, 1148
OESystem::OEOr, 1149
OESystem::OEParameter, 1153
OESystem::OERoles, 1163
OESystem::OEScalarGrid, 1165
OESystem::OESkewGrid, 1169
OESystem::OEUnaryFunction, 1180
OESystem::OEUnaryPredicate, 1183
CreatePredicateCopy
OESystem::OEBinaryPredicate, 1077
OESystem::OEUnaryPredicate, 1183
createringdict.cs
Example Code, 335
Crystal Symmetry, 216
Crystallographic Space Group, 216
Crystallographic Unit Cell, 216
CSV, 26
csv2sdf.cs
Example Code, 310

cyclic membership, 54

D

Daylight, 178, 180
Daylight SMARTS, 180
Daylight SMILES, 178
Debug
    OESystem::OEErrorHandler, 1098
Delete
    OEChem::OEConfBase, 449
    OEChem::OEMolDatabase, 564
DeleteAtom
    OEChem::OEGroupBase, 472
    OEChem::OEMolBase, 554
DeleteAttribute
    OEChem::OECIFCategory, 432
    OEChem::OECIFData, 436
DeleteBond
    OEChem::OEGroupBase, 472
    OEChem::OEMolBase, 554
DeleteCategory
    OEChem::OECIFData, 436
DeleteConf
    OEChem::OEMCMolBase, 536
DeleteConfs
    OEChem::OEMCMolBase, 536
DeleteData
    OESystem::OEBase, 1063
DeleteFeature
    OEBio::OEReceptorCustomConstraints, 1458
DeleteGaussian
    OESystem::OEGaussianContainer, 1115
    OESystem::OEGaussianContainerBase, 1116
DeleteGaussians
    OESystem::OEGaussianContainer, 1115
    OESystem::OEGaussianContainerBase, 1116
DeleteGroup
    OEChem::OEMolBase, 554
DeleteInteraction
    OEBio::OEInteractionHintContainer, 1369
DeleteInteractions
    OEBio::OEInteractionHintContainer, 1369
DeleteInterface
    OESystem::OEInterface, 1135
DeleteOptions
    OESystem::OEOptions, 1253
DeleteParameter
    OESystem::OEInterface, 1135
    OESystem::OEOptions, 1253
DeleteRole
    OESystem::OERoleSet, 1162
DeleteRow
    OEChem::OECIFCategory, 432
    OEChem::OECIFData, 436
DeleteSmarts
    OEBio::OEReceptorConstraintFeature, 1457
DeleteSphere
    OEBio::OEReceptorConstraintFeature, 1456
DeleteStop
    OESystem::OEColorGradientBase, 1090
Describe
    OEBio::OEPlaceHydrogensDetails, 1434
    OEBio::OEPlaceHydrogensMoverNote, 1438
DetermineCategory
    OEBio::OEIridiumData, 1377
Dihedral, 227
DirContents
    OEPlatform::OEFile, 1265
DirPath
    OEPlatform::OEFile, 1265
dockmolecules.cs
    Example Code, 191
dots.cs
    Example Code, 345

E

Elapsed
    OESystem::OECycleTimer, 1095
    OESystem::OEStopwatch, 1173
    OESystem::OEWallTimer, 1186
ElementToGridIdx
    OESystem::OEGridBase, 1123
    OESystem::OESkewGrid, 1169
ElementToSpatialCoord
    OESystem::OEGridBase, 1124
    OESystem::OESkewGrid, 1169
Empty
    OESystem::OEBoundedBuffer, 1082
    OESystem::OEUnboundedBuffer, 1184
eof
    OEChem::oemolistream, 574
    OEPlatform::oestream, 1290
Error
    OESystem::OEErrorHandler, 1099
Example Code
    anisou.cs, 343
    appendringdict.cs, 338
    backbone.cs, 350
    cansmi.cs, 255
    catmols.cs, 244
    cliquealign.cs, 294
    convert.cs, 243
    createringdict.cs, 335
    csv2sdf.cs, 310
    dockmolecules.cs, 191
    dots.cs, 345
    extractringtemplates.cs, 333
    extractscaffold.cs, 266

generate2d.cs, 340
generatemultiplepose.cs, 193
gettitles.cs, 321
libgen.cs, 280
makereceptor.cs, 190
mcs3dalign.cs, 291
minpath.cs, 326
molchunk.cs, 274
molcount.cs, 319
molextract.cs, 248
molgrep.cs, 285
molstats.cs, 323
oecheminfo.cs, 317
parts2mols.cs, 246
printatomnames.cs, 332
printinteractions.cs, 373
proteinprep.cs, 365
randomizeatoms.cs, 254
randomsample.cs, 270
reshist.cs, 351
ringsubset.cs, 264
rmsd.cs, 299
rotorcount.cs, 324
sdf2csv.cs, 307
sdfilter.cs, 314
sdfmodprops.cs, 303
sdfrename.cs, 312
sizefilter.cs, 259
smartsalign.cs, 296
splitmolcomplex.cs, 354
splitmolcomplexfrags.cs, 361
splitmolcomplexlowlevel.cs, 357
stripsalts.cs, 262
unimolrxn.cs, 278
uniqinchi.cs, 252
uniqmol.cs, 251
Exists
OEPlatform::OEFile, 1266
explicit atom degree, 44
explicit hydrogen count, 44
explicit valence, 44
Extend
OEMath::OEBox, 1301
Extract
OEChem::OEEuler, 463
OEChem::OEQuaternion, 612
OEChem::OERotMatrix, 621
OEChem::OETrans, 642
OEChem::OETransBase, 643
OEChem::OETranslation, 644
extractringtemplates.cs
Example Code, 333
extractscaffold.cs
Example Code, 266

F

Factory
OESystem::OEInterface, 1135
Fatal
OESystem::OEErrorHandler, 1099
fd
OEPlatform::oeifstream, 1274
OEPlatform::oeofstream, 1284
file formats, 145
FileName
OEPlatform::OEFile, 1266
FindDupParameters
OESystem::OEOptions, 1254
FirstBit
OESystem::OEBitVector, 1079
Fit
OEMath::OEKernelPLS, 1304
flush
OEChem::oemolostream, 579
OEPlatform::oeostream, 1287
formal charge, 44, 79
FoundInDB
OEBio::OEResidueCategoryData, 1461
FromHexString
OESystem::OEBitVector, 1079
FromString
OEChem::OE3DMolStyle, 387
Full
OESystem::OEBoundedBuffer, 1083
OESystem::OEUnboundedBuffer, 1184
Functor, 217

G

Gasteiger charge, 80
generate2d.cs
Example Code, 340
GenerateHeader
OEChem::OECIFData, 436
generatemultiplepose.cs
Example Code, 193
GeneratePolySeqScheme
OEChem::OECIFData, 437
Get
OEChem::OEMolPropertyFxn, 584
OEChem::OEVectorBindings, 657
OESystem::OEBoundedBuffer, 1083
OESystem::OEInterface, 1135
OESystem::OEOwnedPtr, 1151
OESystem::OEProtectedBuffer, 1159
OESystem::OESharedPtr, 1167
OESystem::OEUnboundedBuffer, 1184
OESystem::OEUnownedPtr, 1185
get
OEPlatform::oeistream, 1279

GetA
OEBio::OECrystalSymmetryParams, 1343
OESystem::OEColor, 1086
GetActive
OEChem::OEMCMolBase, 536
GetAddCarbonSkeleton
OEChem::OE2DRingDictionaryCreatorOptions, 383
GetAddCarbonSkeletonWithBondStereo
OEChem::OE2DRingDictionaryCreatorOptions, 383
GetAddMetadata
OEBio::OEPlaceHydrogensOptions, 1442
GetAddRGroup
OEBio::OESplitMolComplexOptions, 1466
GetAdjustHCount
OEBio::OESplitMolComplexOptions, 1466
GetAliases
OESystem::OEParameter, 1153
GetAllowUnmatchedFragments
OEChem::OEHelmGenerationOptions, 1022
GetAllowUnspecifiedStereo
OEChem::OEHelmGenerationOptions, 1022
GetAlpha
OEBio::OECrystalSymmetryParams, 1343
GetAltAtoms
OEBio::OEAltLocationFactory, 1328
GetAltConfs
OEBio::OEIridiumData, 1377
GetAlteredTag
OEChem::OEUniMolecularRxnOptions, 653
GetAlternateLocation
OEChem::OEResidue, 615
GetAlternateLocation()
OEBio::OEAtomMatchResidueID, 1336
GetAltLoc
OEBio::OEDesignUnit, 1347
GetAltsMustBeCompatible
OEBio::OEPlaceHydrogensOptions, 1443
GetAminoAcidType
OEChem::OEMonomer, 1027
GetAngle
OEBio::OEPlaceHydrogensMoverNote, 1438
GetAngles
OEChem::OEEuler, 463
GetArea
OEMath::OEBox, 1301
OEMath::OESphere, 1306
GetArg
OEPlatform::OEThread, 1271
GetAsDensity
OEBio::OEIridiumData, 1377
GetAssignMapIdx
OEChem::OELibraryGen, 521
GetAssumptions
OEBio::OESplitMolComplexOptions, 1466
GetAtom
OEBio::OEHierChain, 1356
OEBio::OEHierFragment, 1357
OEBio::OEHierResidue, 1358
OEBio::OEHierView, 1360
OEBio::OEInteractionHintFragment, 1373
OEBio::OEReceptorProteinConstraint, 1459
OEChem::OEMolBase, 554
GetAtomColorer
OEChem::OE3DMolStyle, 387
GetAtomicNum
OEChem::OEAtomBase, 399
GetAtomLabelColorer
OEChem::OE3DMolStyle, 388
GetAtomLabelDataTag
OEChem::OE3DMolStyle, 388
GetAtomLabelPrecision
OEChem::OE3DMolStyle, 388
GetAtomLabelText
OEChem::OE3DMolStyle, 388
GetAtomLabelType
OEChem::OE3DMolStyle, 388
GetAtomMapSize
OEBio::OEAltLocationFactory, 1328
GetAtoms
OEBio::OEHierResidue, 1358
OEBio::OEHierView, 1360
OEBio::OEInteractionHintFragment, 1373
OEChem::OEAtomBase, 400
OEChem::OEAtomBondSet, 415
OEChem::OEGroupBase, 472
OEChem::OEMatch, 530
OEChem::OEMatchBase, 531
OEChem::OEMolBase, 555
GetAtomStyle
OEChem::OE3DMolStyle, 388
GetAttributeIndex
OEChem::OECIFCategory, 432
OEChem::OECIFData, 437
GetAttributes
OEChem::OECIFCategory, 432
OEChem::OECIFData, 437
GetAttributeValue
OEChem::OECIFCategory, 432
OEChem::OECIFData, 437
GetAttributeValues
OEChem::OECIFCategory, 432
OEChem::OECIFData, 437
GetAxial
OEChem::OEAtomBase, 400
GetB
OEBio::OECrystalSymmetryParams, 1343

OESystem::OEColor, 1086
GetB0
OEMath::OEKernelPLS, 1304
GetBackboneMatch
OEBio::OESequenceAlignment, 1464
GetBadClashOverlapDistance
OEBio::OEPlaceHydrogensOptions, 1443
GetBaseWriteFxns
OESystem::OEBinaryIOHandler, 1072
OESystem::OEBinaryIOHandlerBase, 1074
GetBeta
OEBio::OECrystalSymmetryParams, 1344
GetBFactor
OEChem::OEResidue, 615
GetBgn
OEChem::OEBondBase, 422
OEChem::OENbrs, 594
GetBgnFragment
OEBio::OEInteractionHint, 1362
GetBgnIdx
OEChem::OEBondBase, 422
GetBigStep
OESystem::OEDots, 1097
GetBinaryIOHandler
OEChem::oemolstreambase, 587
GetBond
OEChem::OEAtomBase, 400
OEChem::OEMolBase, 555
GetBondMapSize
OEBio::OEAltLocationFactory, 1328
GetBonds
OEChem::OEAtomBase, 400
OEChem::OEAtomBondSet, 416
OEChem::OEGroupBase, 472
OEChem::OEMatch, 530
OEChem::OEMatchBase, 531
OEChem::OEMolBase, 555
GetBoolData
OESystem::OEBase, 1064
GetBrief
OESystem::OEInterface, 1135
OESystem::OEOptions, 1254
OESystem::OEParameter, 1153
getbuffer
OEPlatform::oeisstream, 1277
GetBundleCofactors
OEBio::OEMolComplexCategorizer, 1415
OEBio::OEMolComplexCategorizerBase, 1419
GetBValues
OEMath::OEKernelPLS, 1304
GetBypassPredicate
OEBio::OEPlaceHydrogensOptions, 1443
getbyte
OEChem::oemolistream, 574
OEPlatform::oeistream, 1280
GetByteArray
OEChem::oemolostream, 578
GetC
OEBio::OECrystalSymmetryParams, 1344
GetCalculatedGeometries
OEBio::OEInteractionHint, 1364
GetCalculatedGeometry
OEBio::OEInteractionHint, 1363
GetCAlphaMatch
OEBio::OESequenceAlignment, 1464
GetCanon
OEChem::OEParseSmilesOptions, 599
GetCanonicalSmiles
OEChem::OEMonomer, 1027
OEChem::OEMonomerData, 1031
GetCategorizer
OEBio::OESplitMolComplexOptions, 1466
GetCategory
OEBio::OEIridiumData, 1377
OEChem::OECIFData, 438
GetCategoryName
OEChem::OECIFCategory, 432
GetCategoryNames
OEChem::OECIFData, 438
GetCenter
OESystem::OEGaussian, 1110
OESystem::OEGaussianBase, 1112
GetChainID
OEBio::OEAtomMatchResidueID, 1336
OEBio::OEHierChain, 1356
OEBio::OESequenceAlignment, 1464
OEChem::OEResidue, 616
GetChains
OEBio::OEHierView, 1360
GetChelatorType
OEBio::OEChelatorInteractionHint, 1339
GetChi1
OEBio::OERotamer, 1462
GetChi2
OEBio::OERotamer, 1462
GetChi3
OEBio::OERotamer, 1462
GetChi4
OEBio::OERotamer, 1462
GetChildren
OESystem::OEAnnotation, 1060
GetChiral
OEChem::OEInChIOptions, 489
GetClearCoordinates
OEChem::OELibraryGen, 521
OEChem::OEUniMolecularRxn, 650
OEChem::OEUniMolecularRxnOptions, 653
GetClearMapping

OEChem::OERxnAutomapperOptions, 1040
GetClusterScore
OEBio::OEPlaceHydrogensDetails, 1436
GetCode
OEChem::OEMonomer, 1027
OEChem::OEMonomerData, 1030
GetCodes
OEChem::OEMonomer, 1027
OEChem::OEMonomerData, 1030
OEChem::OEMonomerSet, 1035
GetCodeSet
OEChem::OEHelmGenerationOptions, 1022
OEChem::OEHelmGenerationResult, 1023
OEChem::OEHelmParsingResult, 1024
GetCodeSets
OEChem::OEMonomerSet, 1035
GetCollectProteinByChain
OEBio::OESplitMolComplexOptions, 1467
GetColor
OEChem::OEMolStyleColorer, 590
OESystem::OEAnnotation, 1060
OESystem::OEColorStop, 1092
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1112
GetColorAt
OESystem::OEColorGradientBase, 1090
GetColors
OESystem::OEGaussianContainer, 1115
OESystem::OEGaussianContainerBase, 1116
GetColorScheme
OEChem::OEMolStyleColorer, 590
GetColumnName
OEChem::OECSVHeader, 462
GetColumnNames
OEChem::OECSVHeader, 462
GetCombinedFilter
OEBio::OESplitMolComplexOptions, 1467
GetCommandLine
OESystem::OEHeader, 1131
GetComment
OESystem::OEHeader, 1131
GetComponent
OEBio::OEDesignUnit, 1346
GetComponentID
OEBio::OEDesignUnit, 1346
GetComponents
OEBio::OEDesignUnit, 1346
GetComponentsWithin
OEBio::OEDesignUnit, 1346
GetComponentTag
OEBio::OEDesignUnit, 1346
GetComponentType
OEBio::OEInteractionHintFragment, 1373
GetConf
OEChem::OEMCMolBase, 536
GetConfs
OEChem::OEMCMolBase, 537
GetConfTest
OEChem::oemolistream, 571
GetConstraintFeatures
OEBio::OEReceptorCustomConstraints, 1458
GetContourStyle
OEChem::OE3DMolStyle, 388
GetCoords
OEChem::OEConfBase, 450
OEChem::OEMolBase, 555
OESystem::OEGaussianContainer, 1115
OESystem::OEGaussianContainerBase, 1116
GetCounts
OESystem::OEDots, 1097
GetCSVHeader
OEChem::oemolstreambase, 587
GetCurrentAtoms
OEBio::OEAltLocationFactory, 1328
GetCurrentLocations
OEBio::OEAltLocationFactory, 1329
GetCustomConstraints
OEBio::OEReceptor, 1453
GetCXSMILES
OEChem::OEParseSmilesOptions, 599
GetData
OEMath::OEMatrix, 1305
OESystem::OEBase, 1064
OESystem::OEBaseDataT, 1069
OESystem::OEBitVector, 1079
GetDatabaseType
OEChem::OESubSearchDatabase, 631
GetDataItem
OEChem::OECIFCategory, 433
GetDataItems
OEChem::OECIFCategory, 433
OEChem::OECIFData, 438
GetDataIter
OESystem::OEBase, 1064
GetDataType
OEBio::OEDesignUnit, 1346
OEBio::OEInteractionHint, 1364
OEBio::OEInteractionHintFragment, 1374
OEChem::OEMolDatabase, 564
OESystem::OEAnnotation, 1060
OESystem::OEBase, 1064
OESystem::OEBaseData, 1068
OESystem::OEBaseDataT, 1069
OESystem::OEBinaryIOHandler, 1073
OESystem::OEContainer, 1095
OESystem::OEFixedGrid, 1108
OESystem::OEFixedMultiGrid, 1109
OESystem::OEGaussian, 1111

OESystem::OEGaussianBase, 1112
OESystem::OEGaussianContainer, 1115
OESystem::OEGaussianContainerBase, 1117
OESystem::OEGrid, 1119
OESystem::OEGridAbstractBase, 1121
OESystem::OEGridBase, 1124
OESystem::OEMultiGrid, 1146
OESystem::OEScalarGrid, 1166
OESystem::OESkewGrid, 1169
GetDataTypes
OEChem::OEMolStyleColorer, 590
GetDate
OESystem::OEHeader, 1131
GetDefault
OEChem::OEPredicateParameter, 602
OESystem::OEMultiParameter, 1251
OESystem::OETypedParameter, 1177
GetDefaultColor
OEChem::OEMolStyleColorer, 591
GetDefaultReadFxn
OESystem::OEBinaryIOHandler, 1073
OESystem::OEBinaryIOHandlerBase, 1074
GetDefaults
OESystem::OETypedParameter, 1177
GetDefaultsVector
OESystem::OEPODParameter, 1257
GetDefaultWriteFxn
OESystem::OEBinaryIOHandler, 1073
OESystem::OEBinaryIOHandlerBase, 1074
GetDegree
OEChem::OEAtomBase, 400
GetDetail
OESystem::OEInterface, 1135
OESystem::OEOptions, 1254
OESystem::OEParameter, 1154
GetDetails
OEBio::OEInteractionHint, 1364
GetDim
OESystem::OEGridAbstractBase, 1121
OESystem::OEGridBase, 1124
OESystem::OESkewGrid, 1169
GetDimension
OEChem::OEMolBase, 556
GetDist
OEChem::OENbrs, 594
GetDist2
OEChem::OENbrs, 594
GetDoubleData
OESystem::OEBase, 1064
GetDPI
OEBio::OEIridiumData, 1377
GetEnabled
OEBio::OEReceptorConstraintFeature, 1457
OEBio::OEReceptorProteinConstraint, 1459
GetEnd
OEChem::OEBondBase, 423
OEChem::OENbrs, 595
GetEndFragment
OEBio::OEInteractionHint, 1364
GetEndIdx
OEChem::OEBondBase, 423
GetEnergy
OEChem::OEMolBase, 556
GetEntityID
OEChem::OEResidue, 617
GetEntries
OEBio::OEResidueCategoryData, 1461
GetErrorPosition
OEChem::OEHelmParsingResult, 1024
GetExcipients
OEBio::OEIridiumData, 1378
GetExplicitDegree
OEChem::OEAtomBase, 401
GetExplicitHCount
OEChem::OEAtomBase, 401
OEChem::OEHydrogenCounter, 487
GetExplicitHydrogens
OEChem::OELibraryGen, 522
GetExplicitValence
OEChem::OEAtomBase, 401
GetExponent
OESystem::OEExponentColorGradient, 1104
GetExpr
OEChem::OEQBase, 605
GetExtChainID
OEChem::OEResidue, 616
GetExtend
OEBio::OESequenceAlignment, 1465
GetExtendedSaltBridgeDistance
OEBio::OEPerceiveInteractionOptions, 1421
GetExtendMappedRegions
OEChem::OERxnAutomapperOptions, 1040
GetFailed
OEChem::OEMMFFTypeResult, 550
GetFeatureName
OEBio::OEReceptorConstraintFeature, 1457
GetFeatureSpheres
OEBio::OEReceptorConstraintFeature, 1456
GetFileName
OEChem::oemolstreambase, 587
GetFixedHLayer
OEChem::OEInChIOptions, 489
GetFixValence
OEChem::OEUniMolecularRxnOptions, 654
GetFlavor, 21
OEChem::OEMolDatabase, 564
OEChem::OEMolDatabaseSaveOptions, 569
OEChem::oemolstreambase, 587

GetFlipBias
OEBio::OEPlaceHydrogensDetails, 1436
OEBio::OEPlaceHydrogensOptions, 1443
GetFlipBiasScale
OEBio::OEPlaceHydrogensOptions, 1444
GetFlippedBestClusterScore
OEBio::OEPlaceHydrogensMoverNote, 1438
GetFlipPredicate
OEBio::OEPlaceHydrogensOptions, 1444
GetFloatData
OESystem::OEBase, 1065
GetFormalCharge
OEChem::OEAtomBase, 401
GetFormat, 14
OEChem::OEMolDatabase, 564
OEChem::OEMolDatabaseSaveOptions, 569
OEChem::oemolstreambase, 587
GetFragment
OEBio::OEInteractionHint, 1365
GetFragmentNumber
OEBio::OEAtomMatchResidueID, 1336
OEBio::OEHierFragment, 1357
OEChem::OEResidue, 616
GetFragments
OEBio::OEHierChain, 1356
OEBio::OEHierView, 1360
GetG
OESystem::OEColor, 1086
GetGamma
OEBio::OECrystalSymmetryParams, 1344
GetGap
OEBio::OESequenceAlignment, 1465
GetGaussians
OESystem::OEGaussianContainer, 1115
OESystem::OEGaussianContainerBase, 1117
GetGridContourColorer
OEChem::OE3DMolStyle, 388
GetGridType
OEChem::OE3DMolStyle, 388
GetGroup
OEBio::OEAltLocationFactory, 1329
OEChem::OEMolBase, 556
GetGroupCount
OEBio::OEAltGroup, 1325
OEBio::OEAltLocationFactory, 1329
GetGroupDefinition
OEChem::OEGroupBase, 473
GetGroupID
OEBio::OEAltGroup, 1325
OEBio::OEAltLocation, 1326
GetGroups
OEBio::OEAltLocationFactory, 1329
OEChem::OEMolBase, 556
GetGroupType
OEChem::OEGroupBase, 473
Getgz
OEChem::oemolstreambase, 588
GetHandlerImpl
OESystem::OEErrorHandler, 1099
GetHasDefault
OESystem::OEParameter, 1154
GetHasMTZ
OEBio::OEIridiumData, 1378
GetHasValue
OESystem::OEParameter, 1154
GetHBondType
OEBio::OEHBondInteractionHint, 1355
GetHistory
OESystem::OEHeader, 1131
GetHvyDegree
OEChem::OEAtomBase, 401
GetHvyValence
OEChem::OEAtomBase, 401
GetHyb
OEChem::OEAtomBase, 401
GetHydrogenConversions
OEChem::OEUniMolecularRxnOptions, 654
GetHydrogens
OEChem::OEInChIOptions, 489
GetHydrogenVisibility
OEChem::OE3DMolStyle, 389
GetIDs
OEBio::OEDesignUnit, 1346
GetIdx
OEChem::OEAtomBase, 402
OEChem::OEBondBase, 423
OEChem::OEConfBase, 450
OEChem::OEMolDatabase, 565
GetIdxFileName
OEChem::OEMolDatabaseSaveOptions, 569
GetIdxs
OEChem::OEMolDatabase, 565
GetIgnoreHydrogens
OEChem::OEConfRMSDOptions, 456
GetIgnoreTerminalHeavy
OEChem::OEConfRMSDOptions, 456
GetIllegalRanges
OESystem::OEParameter, 1154
GetIllegalValues
OESystem::OEParameter, 1154
GetImplicitHCount
OEChem::OEAtomBase, 402
OEChem::OEHydrogenCounter, 487
GetInFile
OEChem::OERefInputAppOptions, 613
OEChem::OESimpleAppOptions, 624
GetInnerContourLevel
OEBio::OEReceptor, 1453

GetInnerContourVolume
OEBio::OEReceptor, 1453
GetInsertCode
OEBio::OEAtomMatchResidueID, 1336
OEChem::OEResidue, 616
GetIntData
OESystem::OEBase, 1065
GetInteractions
OEBio::OEInteractionHintContainer, 1369
GetInteractionStyle
OEChem::OE3DMolStyle, 389
GetInteractionType
OEBio::OEInteractionHint, 1365
GetInterface
OESystem::OEHeader, 1131
OESystem::OEInterface, 1135
GetInterfaces
OESystem::OEInterface, 1136
GetIntType
OEChem::OEAtomBase, 402
OEChem::OEBondBase, 423
GetInverse
OEChem::OEEuler, 464
OEChem::OEQuaternion, 612
OEChem::OERotMatrix, 621
OEChem::OETrans, 642
OEChem::OETransBase, 643
OEChem::OETranslation, 644
GetIridiumData
OEBio::OEStructureQuality, 1474
GetIrrationalRfree
OEBio::OEIridiumData, 1378
GetIsList
OESystem::OEParameter, 1154
GetIsotope
OEChem::OEAtomBase, 402
GetIStream
OEChem::oemolithread, 576
GetKeepTitle
OEChem::OECreateSubSearchDatabaseOptions, 459
GetKeyless
OESystem::OEParameter, 1154
GetLabel
OEBio::OEPlaceHydrogensMoverNote, 1439
GetLegalRanges
OESystem::OEParameter, 1154
GetLegalValues
OESystem::OEParameter, 1155
GetLevel
OESystem::OEErrorHandler, 1099
OESystem::OEErrorHandlerImplBase, 1102
GetLigand
OEBio::OEDesignUnit, 1346
GetLigandDensity
OEBio::OEIridiumData, 1378
GetLigandFilter
OEBio::OESplitMolComplexOptions, 1467
GetLigandSubSearch
OEBio::OEMolComplexCategorizer, 1415
getline
OEChem::oemolistream, 574
OEPlatform::oeistream, 1280
GetList
OESystem::OEInterface, 1136
GetLocation
OEBio::OEAltLocationFactory, 1329
GetLocationCodes
OEBio::OEAltLocationFactory, 1330
GetLocationCount
OEBio::OEAltGroup, 1325
OEBio::OEAltLocation, 1327
GetLocationID
OEBio::OEAltLocation, 1327
GetLocations
OEBio::OEAltGroup, 1325
GetMapIdx
OEChem::OEAtomBase, 402
GetMappingMethod
OEChem::OERxnAutomapperOptions, 1040
GetMatchIndices
OEChem::OESubSearchDatabase, 631
OEChem::OESubSearchResult, 636
GetMatchTitles
OEChem::OESubSearchDatabase, 631
GetMaxAcceptorAngle
OEBio::OEPerceiveInteractionOptions, 1422
GetMaxAtomIdx
OEChem::OEMolBase, 557
GetMaxAtomPairDistance
OEBio::OEPerceiveInteractionOptions, 1422
GetMaxBindingSiteDist
OEBio::OESplitMolComplexOptions, 1467
GetMaxBondIdx
OEChem::OEMolBase, 557
GetMaxCationPiAngle
OEBio::OEPerceiveInteractionOptions, 1422
GetMaxCationPiDistance
OEBio::OEPerceiveInteractionOptions, 1422
GetMaxChargeAidedHBondDistance
OEBio::OEPerceiveInteractionOptions, 1422
GetMaxColor
OESystem::OEColorGradientBase, 1090
GetMaxConfIdx
OEChem::OEMCMolBase, 537
GetMaxContactFraction
OEBio::OEPerceiveInteractionOptions, 1422
GetMaxCovalentFraction

OEBio::OEPerceiveInteractionOptions, 1422
GetMaxDonorAngle
OEBio::OEPerceiveInteractionOptions, 1422
GetMaxHalogenAngle
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxHalogenBondDistance
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxHBondDistance
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxHeavyAtom
OEBio::OELigandInteractionHintComponent, 1413
GetMaxLigAtoms
OEBio::OEMolComplexCategorizer, 1415
GetMaxLigResidues
OEBio::OEMolComplexCategorizer, 1416
GetMaxMatches
OEChem::OECliqueSearch, 445
OEChem::OEConfRMSDOptions, 456
OEChem::OEMCSSearch, 547
OEChem::OESubSearch, 627
OEChem::OESubSearchQuery, 635
OEChem::OEUniMolecularRxnOptions, 654
GetMaxMetalBondDistance
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxMolIdx
OEChem::OEMolDatabase, 565
GetMaxNonIdealAcceptorAngle
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxNonIdealDonorAngle
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxNonIdealHBondDistance
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxPeptideResidues
OEBio::OEMolComplexCategorizer, 1416
GetMaxPiStackAngle
OEBio::OEPerceiveInteractionOptions, 1423
GetMaxPiStackDistance
OEBio::OEPerceiveInteractionOptions, 1424
GetMaxProteinAtoms
OEBio::OEMolComplexCategorizer, 1416
GetMaxProteinIntraDistance
OEBio::OEPerceiveInteractionOptions, 1424
GetMaxSaltBridgeDistance
OEBio::OEPerceiveInteractionOptions, 1424
GetMaxSeqIdx
OEBio::OESequenceAlignment, 1465
GetMaxSubstateCutoff
OEBio::OEPlaceHydrogensOptions, 1444
GetMaxSurfaceWaterDist
OEBio::OESplitMolComplexOptions, 1467
GetMaxTStackAngle
OEBio::OEPerceiveInteractionOptions, 1424
GetMaxTStackDistance
OEBio::OEPerceiveInteractionOptions, 1424
GetMaxUnpairedAngle
OEBio::OEPerceiveInteractionOptions, 1424
GetMaxUnpairedDistance
OEBio::OEPerceiveInteractionOptions, 1424
GetMaxValue
OESystem::OEColorGradientBase, 1091
GetMCMol
OEChem::OEConfBase, 450
GetMCMolBases
OEChem::oemolistream, 572
GetMCMolTitle
OEChem::OEMCMolBase, 537
GetMCSExhaustiveMode
OEChem::OERxnAutomapperOptions, 1040
GetMetal
OEBio::OEDesignUnit, 1346
GetMethod
OEBio::OESequenceAlignment, 1465
GetMid
OESystem::OEGridAbstractBase, 1121
OESystem::OEGridBase, 1124
OESystem::OESkewGrid, 1169
GetMinAtoms
OEChem::OECliqueSearch, 445
OEChem::OEMCSSearch, 547
GetMinColor
OESystem::OEColorGradientBase, 1091
GetMinContactFraction
OEBio::OEPerceiveInteractionOptions, 1424
GetMinCovalentFraction
OEBio::OEPerceiveInteractionOptions, 1425
GetMinHBondContactFraction
OEBio::OEPerceiveInteractionOptions, 1425
GetMinLigAtoms
OEBio::OEMolComplexCategorizer, 1416
GetMinStackOffset
OEBio::OEPerceiveInteractionOptions, 1425
GetMinUnmatchedAtomsRatio
OEChem::OEHelmGenerationOptions, 1022
GetMinValue
OESystem::OEColorGradientBase, 1091
GetMMFFType
OEChem::OEMMFFTypeResult, 550
GetModelNumber
OEChem::OEResidue, 616
GetMol
OEChem::oemolthreadbase, 592
GetMolBases
OEChem::oemolistream, 572
GetMolecule
OEBio::OEInteractionHintContainer, 1370
OEChem::OEMolDatabase, 565
OEChem::OESubSearchDatabase, 631

GetMolecules
OEBio::OEInteractionHintContainer, 1371
GetMoleculeString
OEChem::OEMolDatabase, 565
GetMolStream
OEChem::oemolithread, 576
OEChem::oemolothread, 580
GetMonomer
OEChem::OEMonomerSet, 1035
GetMonomers
OEChem::OEMonomerSet, 1035
GetMonomerType
OEChem::OEMonomer, 1027
OEChem::OEMonomerData, 1030
GetMoverClass
OEBio::OEPlaceHydrogensMoverNote, 1439
GetMoverNotes
OEBio::OEPlaceHydrogensDetails, 1436
GetName
OEBio::OEAtomMatchResidueID, 1335
OEBio::OEInteractionHintComponentTypeBase, 1367
OEBio::OEInteractionHintTypeBase, 1375
OEBio::OEReceptorProteinConstraint, 1459
OEChem::OEAtomBase, 403
OEChem::OECIFData, 438
OEChem::OEMonomer, 1028
OEChem::OEMonomerData, 1030
OEChem::OEResidue, 617
OEChem::OESubSearchScreenTypeBase, 639
OESystem::OEInterface, 1136
OESystem::OEOptions, 1254
OESystem::OEParameter, 1155
OESystem::OERole, 1162
GetNbr
OEChem::OEBondBase, 423
GetNbrs
OEChem::OENearestNbrs, 595
GetNegativeImageGrid
OEBio::OEReceptor, 1453
GetNextToken
OEChem::OETokenizerBase, 640
GetNoFlipPredicate
OEBio::OEPlaceHydrogensOptions, 1444
GetNonPolymerPseudoAtoms
OEChem::OEInChIOptions, 489
GetNucleicAcid
OEBio::OEDesignUnit, 1346
GetNumAttributes
OEChem::OECIFCategory, 433
GetNumCategories
OEChem::OECIFData, 438
GetNumCategoryAttributes
OEChem::OECIFData, 438
GetNumCategoryRows
OEChem::OECIFData, 438
GetNumClusters
OEBio::OEPlaceHydrogensDetails, 1436
GetNumColors
OEChem::OEMolStyleColorer, 591
GetNumColumns
OEChem::OECSVHeader, 462
OEMath::OEMatrix, 1306
GetNumericAttributeValue
OEChem::OECIFCategory, 433
OEChem::OECIFData, 438
GetNumericAttributeValues
OEChem::OECIFCategory, 433
OEChem::OECIFData, 438
GetNumFeaturesUsed
OEMath::OEKernelPLS, 1304
GetNumFlips
OEBio::OEPlaceHydrogensDetails, 1437
GetNumGrids
OESystem::OEFixedMultiGrid, 1109
GetNumHydrogensStandardized
OEBio::OEPlaceHydrogensDetails, 1437
GetNumMovers
OEBio::OEPlaceHydrogensDetails, 1437
GetNumProcessors
OEChem::OECreateSubSearchDatabaseOptions, 459
GetNumRows
OEChem::OECIFCategory, 433
OEMath::OEMatrix, 1306
GetOccupancy
OEChem::OEResidue, 617
GetOEGraphMols
OEChem::oemolistream, 572
GetOERelease
OESystem::OEHeader, 1131
GetOEResidue
OEBio::OEHierChain, 1357
OEBio::OEHierFragment, 1357
OEBio::OEHierResidue, 1358
GetOEVersion
OESystem::OEHeader, 1132
GetOption
OESystem::OEOptions, 1254
GetOptions
OEChem::OEConfRMSD, 454
OEChem::OERxnAutomapper, 1039
OEChem::OEUniMolecularRxn, 650
OESystem::OEOptions, 1254
GetOrder
OEChem::OEBondBase, 424
GetOrderPriority
OESystem::OEInterface, 1136

OESystem::OEOptions, 1254
OESystem::OEParameter, 1155
GetOriginalBestClusterScore
OEBio::OEPlaceHydrogensMoverNote, 1439
GetOStream
OEChem::oemolothread, 580
GetOtherFilter
OEBio::OESplitMolComplexOptions, 1467
GetOuterContourLevel
OEBio::OEReceptor, 1453
GetOuterContourVolume
OEBio::OEReceptor, 1453
GetOutFile
OEChem::OERefInputAppOptions, 614
OEChem::OESimpleAppOptions, 624
GetPackedColor
OESystem::OEColor, 1086
GetPackingResidues
OEBio::OEDesignUnit, 1346
OEBio::OEIridiumData, 1378
GetParameter
OESystem::OEInterface, 1136
OESystem::OEOptions, 1255
GetParameters
OESystem::OEInterface, 1136
OESystem::OEOptions, 1255
GetParent
OEChem::OEAtomBase, 403
OEChem::OEAtomBondSet, 416
OEChem::OEBondBase, 424
OEChem::OEGroupBase, 473
GetParentAtom
OEChem::OEQAtomBase, 604
GetParentBond
OEChem::OEQBondBase, 606
GetPartialCharge
OEChem::OEAtomBase, 403
GetPartOccupancyLigand
OEBio::OEIridiumData, 1378
GetPartOccupancyProtein
OEBio::OEIridiumData, 1378
GetPattern
OEChem::OECliqueSearch, 446
OEChem::OEMCSSearch, 547
OEChem::OESubSearch, 627
GetPatternAtoms
OEChem::OEMatchBase, 532
GetPatternBonds
OEChem::OEMatchBase, 532
GetPDBMetaData
OEBio::OEDesignUnit, 1346
GetPerceiveCell
OEChem::OECIFOptions, 442
GetPerceiveChemComp
OEChem::OECIFOptions, 442
GetPerceiveEntity
OEChem::OECIFOptions, 442
GetPerceiveExperimental
OEChem::OECIFOptions, 443
GetPerceiveOther
OEChem::OECIFOptions, 443
GetPerceiveStruct
OEChem::OECIFOptions, 442
GetPerceiveSymmetry
OEChem::OECIFOptions, 442
GetPhi
OEBio::OERamachandranAnalysis, 1450
OEChem::OEEuler, 464
GetPlatform
OESystem::OEHeader, 1132
GetPolymerType
OEChem::OEMonomer, 1028
OEChem::OEMonomerData, 1031
GetPossibleCovalent
OEBio::OEIridiumData, 1378
GetPrefactor
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
GetPrefactors
OESystem::OEGaussianContainer, 1115
OESystem::OEGaussianContainerBase, 1117
GetPrettyLegalValues
OESystem::OEParameter, 1155
GetPrimaryCodeSet
OEChem::OEMonomerSet, 1035
GetProbability
OEBio::OERotamer, 1462
GetProducts
OEChem::OELibraryGen, 522
GetProgram
OESystem::OEHeader, 1132
GetPromoteUnknowns
OEBio::OESplitMolComplexOptions, 1468
GetProperty
OEChem::OEMolPropertyList, 585
GetPropertyName
OEChem::OEMolStyleColorer, 591
GetProtein
OEBio::OEDesignUnit, 1346
GetProteinColorer
OEChem::OE3DMolStyle, 389
GetProteinConstraints
OEBio::OEReceptor, 1453
GetProteinFilter
OEBio::OESplitMolComplexOptions, 1468
GetProteinStyle
OEChem::OE3DMolStyle, 389
GetPsi

OEBio::OERamachandranAnalysis, 1450
OEChem::OEEuler, 464
GetPtr
OEChem::OEConstCoords, 458
OEChem::OEMutableCoords, 594
GetQAtoms
OEChem::OEQAtomBase, 604
OEChem::OEQMol, 609
OEChem::OEQMolBase, 611
GetQBonds
OEChem::OEQAtomBase, 605
OEChem::OEQMol, 609
OEChem::OEQMolBase, 611
GetQuaternion
OEChem::OEQuaternion, 612
GetQuiet
OEChem::OEParseSmilesOptions, 599
GetR
OESystem::OEColor, 1085
GetRacemicStereo
OEChem::OEInChIOptions, 490
GetRad
OEMath::OESphere, 1307
GetRadius
OEChem::OEAtomBase, 403
GetRamaCategory
OEBio::OERamachandranAnalysis, 1451
GetRamaScore
OEBio::OERamachandranAnalysis, 1451
GetRamaType
OEBio::OERamachandranAnalysis, 1451
GetReadFailureCt
OEChem::oemolistream, 572
GetReadFxns
OESystem::OEBinaryIOHandler, 1073
OESystem::OEBinaryIOHandlerBase, 1074
GetReceptor
OEBio::OEDesignUnit, 1346
GetReconnectedMetals
OEChem::OEInChIOptions, 490
GetReentrant
OESystem::OESkewGrid, 1169
GetReferenceCoords
OEBio::OEPlaceHydrogensMoverNote, 1439
GetRefFile
OEChem::OERefInputAppOptions, 614
GetRelativeStereo
OEChem::OEInChIOptions, 490
GetRemoveUnmappedFragments
OEChem::OELibraryGen, 522
GetRenderHint
OESystem::OEAnnotation, 1060
GetRequired
OESystem::OEParameter, 1155
GetResidue
OEBio::OEHierChain, 1357
OEBio::OEHierFragment, 1358
OEBio::OEHierView, 1360
GetResidueCategoryData
OEBio::OEMolComplexCategorizer, 1416
OEBio::OEMolComplexCategorizerBase, 1419
GetResidueIndex
OEBio::OERotamer, 1462
GetResidueName
OEBio::OEHierResidue, 1359
GetResidueNumber
OEBio::OEAtomMatchResidueID, 1335
OEBio::OEHierResidue, 1359
OEChem::OEResidue, 617
GetResidues
OEBio::OEHierFragment, 1358
OEBio::OEHierView, 1360
GetResolution
OEBio::OEIridiumData, 1379
GetRetainBondStereo
OEChem::OEExtractRingTemplateOptions, 466
GetRetainExistingBuiltInTemplates
OEChem::OE2DRingDictionaryCreatorOptions, 384
GetRetainExistingTemplates
OEChem::OE2DRingDictionaryCreatorOptions, 384
OEChem::OEExtractRingTemplateOptions, 466
GetReturn
OEPlatform::OEThread, 1271
GetReturnCode
OEChem::OEHelmGenerationResult, 1023
OEChem::OEHelmParsingResult, 1024
OEChem::OEMonomerValidationResult, 1038
GetRFree
OEBio::OEIridiumData, 1379
GetRGBA
OESystem::OEColor, 1086
GetRoles
OESystem::OERoleSet, 1163
GetRotMatrix
OEChem::OERotMatrix, 621
GetRow
OEChem::OECIFCategory, 433
OEChem::OECIFData, 439
OEChem::OEMolDatabase, 566
GetRowIndices
OEChem::OECIFCategory, 434
OEChem::OECIFData, 439
GetRows
OEChem::OECIFCategory, 433
OEChem::OECIFData, 439
GetRxnRole

OEChem::OEAtomBase, 403
GetSaveRange
OEChem::OECliqueSearch, 446
GetScore
OEBio::OEPlaceHydrogensMoverNote, 1439
OEBio::OESequenceAlignment, 1465
GetScreenType
OEChem::OECreateSubSearchDatabaseOptions, 460
OEChem::OESubSearchDatabase, 632
OEChem::OESubSearchScreen, 638
OEChem::OESubSearchScreenTypeBase, 639
GetScreenTypeString
OEChem::OESubSearchScreenTypeBase, 639
GetSearchStatus
OEChem::OESubSearchResult, 636
GetSecondaryStructure
OEChem::OEResidue, 617
GetSeed
OESystem::OERandom, 1160
GetSeparateResidues
OEBio::OESplitMolComplexOptions, 1468
GetSequenceID
OEChem::OEResidue, 617
GetSequenceIdentity
OEBio::OESequenceAlignment, 1465
GetSequenceSimilarity
OEBio::OESequenceAlignment, 1465
GetSerialNumber
OEChem::OEResidue, 617
GetSetting
OEChem::OEMolParameter, 581
OEChem::OEPredicateParameter, 603
OESystem::OEMultiParameter, 1251
OESystem::OETypedParameter, 1177
GetSettings
OESystem::OETypedParameter, 1177
GetSettingsVector
OESystem::OEPODParameter, 1257
GetSiteResidues
OEBio::OEDesignUnit, 1346
GetSize
OEChem::OEMolPropertyList, 585
OEChem::OESubSearchScreenTypeBase, 640
OESystem::OEBaseData, 1068
OESystem::OEBaseDataT, 1069
OESystem::OEBitVector, 1079
OESystem::OEGridAbstractBase, 1121
OESystem::OEGridBase, 1124
OESystem::OESkewGrid, 1170
GetSmallStep
OESystem::OEDots, 1097
GetSmarts
OEBio::OEReceptorConstraintFeature, 1456
GetSmiles
OEChem::OEMonomer, 1028
OEChem::OEMonomerData, 1031
GetSolvent
OEBio::OEDesignUnit, 1346
GetSortByBitCounts
OEChem::OECreateSubSearchDatabaseOptions, 460
GetSourceMol
OEBio::OEAltLocationFactory, 1330
GetSpaceGroup
OEBio::OECrystalSymmetryParams, 1344
OESystem::OESkewGrid, 1170
GetSpacing
OESystem::OEGridAbstractBase, 1121
OESystem::OEGridBase, 1124
OESystem::OESkewGrid, 1170
GetSplitCovalent
OEBio::OESplitMolComplexOptions, 1468
GetSplitCovalentCofactors
OEBio::OESplitMolComplexOptions, 1468
GetStackSize
OEPlatform::OEThread, 1272
GetStandardizeBondLen
OEBio::OEPlaceHydrogensOptions, 1445
GetStartingMaterial
OEChem::OELibraryGen, 522
GetState
OEBio::OEAltLocationFactory, 1330
GetStateIdx
OEBio::OEPlaceHydrogensMoverNote, 1440
GetStateName
OEBio::OEPlaceHydrogensMoverNote, 1440
GetStereo
OEChem::OEAtomBase, 404
OEChem::OEBondBase, 424
OEChem::OEInChIOptions, 490
GetStops
OESystem::OEColorGradientBase, 1091
GetStrict
OEChem::OEParseSmilesOptions, 599
OESystem::OEErrorHandler, 1099
OESystem::OEErrorHandlerImplBase, 1102
GetStrictMapping
OEChem::OERxnAutomapperOptions, 1040
GetStrictSmirks
OEChem::OEUniMolecularRxnOptions, 654
GetString
OEChem::OE3DMolStyle, 389
OEChem::oemolothread, 580
GetStringData
OESystem::OEBase, 1065
GetStringDefault
OESystem::OEParameter, 1155

GetStringDefaults
OESystem::OEParameter, 1155
GetStringValue
OESystem::OEParameter, 1155
GetStringValues
OESystem::OEParameter, 1156
GetStructureQuality
OEBio::OEDesignUnit, 1346
GetSubChainID
OEChem::OEResidue, 616
GetSurfaceAlpha
OEChem::OE3DMolStyle, 389
GetSurfaceColorer
OEChem::OE3DMolStyle, 389
GetSurfaceStyle
OEChem::OE3DMolStyle, 389
GetSurfaceType
OEChem::OE3DMolStyle, 389
GetSymmetryClass
OEChem::OEAtomBase, 404
GetTag
OEChem::OESDDataPair, 622
OESystem::OEBaseData, 1068
GetTaggedComponents
OEBio::OEDesignUnit, 1346
GetTags
OEBio::OEDesignUnit, 1347
GetTarget
OEBio::OEDesignUnit, 1347
GetTargetAtoms
OEChem::OEMatchBase, 532
GetTargetBonds
OEChem::OEMatchBase, 532
GetTargetComponentID
OEBio::OEDesignUnit, 1347
GetTargetComponentTag
OEBio::OEDesignUnit, 1347
GetTargetMask
OEBio::OEReceptor, 1454
GetTargetPred
OEBio::OEReceptor, 1454
GetText
OESystem::OEColor, 1086
GetTheta
OEChem::OEEuler, 464
GetTimeout
OEChem::OEInChIOptions, 490
GetTitle
OEBio::OEDesignUnit, 1347
OEBio::OEInteractionHintContainer, 1371
OEChem::OEConfBase, 450
OEChem::OEMolBase, 557
OEChem::OEMolDatabase, 566
OEChem::OESubSearchDatabase, 632
OESystem::OEAnnotation, 1060
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1124
OESystem::OESkewGrid, 1170
GetTitles
OEChem::OEMolDatabase, 566
gettitles.cs
Example Code, 321
GetTitleSeparator
OEChem::OELibraryGen, 523
gettoken
OEPlatform::oeistream, 1280
GetTorsion
OEChem::OEConfBase, 451
GetTotalHCount
OEChem::OEAtomBase, 404
GetTransform
OEBio::OEDesignUnit, 1346
OEChem::OEConfBase, 451
GetTransforms
OEChem::OETrans, 642
GetTranslation
OEChem::OETranslation, 645
GetType
OEBio::OEReceptorProteinConstraint, 1459
OEChem::OEAtomBase, 404
OEChem::OEBondBase, 424
OEChem::OEExprBase, 465
OEChem::OELingoSim, 527
OESystem::OEAnnotation, 1060
OESystem::OEScalarGrid, 1166
OESystem::OESkewGrid, 1170
GetTypedParameter
OESystem::OEInterface, 1137
GetU11
OEChem::OEAnisoUij, 398
GetU12
OEChem::OEAnisoUij, 398
GetU13
OEChem::OEAnisoUij, 398
GetU22
OEChem::OEAnisoUij, 398
GetU23
OEChem::OEAnisoUij, 398
GetU33
OEChem::OEAnisoUij, 398
GetUndefinedStereo
OEChem::OEInChIOptions, 490
GetUnitCell
OESystem::OESkewGrid, 1170
GetUnmappedFragments
OEChem::OEHelmGenerationResult, 1024
GetValence
OEChem::OEAtomBase, 405

GetValenceCorrection
OEChem::OELibraryGen, 523
GetValidateKekule
OEChem::OELibraryGen, 523
OEChem::OEUniMolecularRxn, 651
OEChem::OEUniMolecularRxnOptions, 655
GetValue
OEChem::OEMolParameter, 582
OEChem::OEPredicateParameter, 603
OEChem::OESDDataPair, 622
OEPlatform::OEHalfFloat, 1267
OEPlatform::OEOnce, 1270
OESystem::OEColorStop, 1092
OESystem::OEMultiParameter, 1251
OESystem::OETypedParameter, 1177
GetValues
OEChem::OEMolStyleColorer, 591
OESystem::OEFixedGrid, 1108
OESystem::OEFixedMultiGrid, 1109
OESystem::OESkewGrid, 1170
OESystem::OETypedParameter, 1177
GetValuesVector
OESystem::OEPODParameter, 1257
GetVersion
OEChem::OEMonomerSet, 1035
OEChem::OESubSearchScreenTypeBase, 640
OESystem::OEHeader, 1132
GetVersionString
OEChem::OESubSearchScreenTypeBase, 640
GetVisibility
OESystem::OEParameter, 1156
GetVolume
OEMath::OEBox, 1301
OEMath::OESphere, 1307
GetWarning
OEChem::OEHelmGenerationResult, 1024
OEChem::OEHelmParsingResult, 1025
OEChem::OEMonomerValidationResult, 1038
GetWarnNoLigand
OEBio::OESplitMolComplexOptions, 1468
GetWaterFilter
OEBio::OESplitMolComplexOptions, 1468
GetWaterProcessing
OEBio::OEPlaceHydrogensOptions, 1445
GetWeight
OEBio::OEInteractionHint, 1366
GetWidth
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
GetWidths
OESystem::OEGaussianContainer, 1115
OESystem::OEGaussianContainerBase, 1117
GetWriteFxns
OESystem::OEBinaryIOHandler, 1073
OESystem::OEBinaryIOHandlerBase, 1074
GetWriteIdx
OEChem::OEMolDatabaseSaveOptions, 569
GetX
OEMath::OESphere, 1307
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
OESystem::OEGridBase, 1124
GetXDim
OEMath::OEBox, 1301
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1125
OESystem::OESkewGrid, 1170
GetXIdx
OESystem::OEGridBase, 1125
GetXInc
OESystem::OEGridBase, 1125
GetXMax
OEMath::OEBox, 1302
OESystem::OEGridBase, 1125
GetXMid
OEMath::OEBox, 1302
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1125
OESystem::OESkewGrid, 1170
GetXMin
OEMath::OEBox, 1302
OESystem::OEGridBase, 1125
GetY
OEMath::OESphere, 1307
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
OESystem::OEGridBase, 1125
GetYDim
OEMath::OEBox, 1301
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1126
OESystem::OESkewGrid, 1171
GetYIdx
OESystem::OEGridBase, 1126
GetYInc
OESystem::OEGridBase, 1126
GetYMax
OEMath::OEBox, 1302
OESystem::OEGridBase, 1126
GetYMid
OEMath::OEBox, 1302
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1126
OESystem::OESkewGrid, 1171
GetYMin
OEMath::OEBox, 1303
OESystem::OEGridBase, 1126
GetZ

OEMath::OESphere, 1307
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
OESystem::OEGridBase, 1126
GetZDim
OEMath::OEBox, 1302
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1127
OESystem::OESkewGrid, 1171
GetZIdx
OESystem::OEGridBase, 1127
GetZInc
OESystem::OEGridBase, 1127
GetZMax
OEMath::OEBox, 1302
OESystem::OEGridBase, 1127
GetZMid
OEMath::OEBox, 1302
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1127
OESystem::OESkewGrid, 1171
GetZMin
OEMath::OEBox, 1303
OESystem::OEGridBase, 1127
GetZValue
OEBio::OECrystalSymmetryParams, 1344
GONNET, 216
GridIdxToElement
OESystem::OEGridBase, 1127
GridIdxToSpatialCoord
OESystem::OEGridBase, 1128

H

Has
OEChem::OEMolPropertyFxn, 585
OESystem::OEInterface, 1137
HasAlphaBetaUnsat
OEChem::OEAtomBase, 405
HasAltLoc
OEBio::OEDesignUnit, 1347
HasAtom
OEBio::OEInteractionHintFragment, 1374
OEChem::OEAtomBondSet, 415
OEChem::OEGroupBase, 473
HasAtomColorer
OEChem::OE3DMolStyle, 390
HasAtomLabelColorer
OEChem::OE3DMolStyle, 390
HasAtomLabelDataTag
OEChem::OE3DMolStyle, 390
HasAtomLabelText
OEChem::OE3DMolStyle, 390
HasAtomLabelType
OEChem::OE3DMolStyle, 390
HasAtomStyle
OEChem::OE3DMolStyle, 390
HasAttribute
OEChem::OECIFCategory, 434
OEChem::OECIFData, 439
HasBadClash
OEBio::OEPlaceHydrogensMoverNote, 1440
HasBond
OEChem::OEAtomBondSet, 415
OEChem::OEGroupBase, 473
HasCalculatedGeometry
OEBio::OEInteractionHint, 1362
HasCategoryName
OEChem::OECIFData, 439
HasCode
OEChem::OEMonomer, 1028
OEChem::OEMonomerData, 1031
HasCodeInCodeSet
OEChem::OEMonomerSet, 1036
HasCodeSet
OEChem::OEMonomerSet, 1036
HasCompareMols
OEChem::OEConfTestBase, 457
OEChem::OEDefaultConfTest, 463
HasComponent
OEBio::OEDesignUnit, 1347
HasContourStyle
OEChem::OE3DMolStyle, 390
HasCustomConstraints
OEBio::OEReceptor, 1454
HasData
OESystem::OEBase, 1065
HasDupParameters
OESystem::OEOptions, 1255
HasFlippedBadClusterClash
OEBio::OEPlaceHydrogensMoverNote, 1440
HasGridContourColorer
OEChem::OE3DMolStyle, 390
HasGridType
OEChem::OE3DMolStyle, 390
HasHydrogenVisibility
OEChem::OE3DMolStyle, 391
HasInnerContourLevel
OEBio::OEReceptor, 1454
HasInteraction
OEBio::OEInteractionHintContainer, 1371
HasInteractionStyle
OEChem::OE3DMolStyle, 391
HasIridiumData
OEBio::OEStructureQuality, 1474
HasLigand
OEBio::OEDesignUnit, 1347
HasMetal
OEBio::OEDesignUnit, 1347

HasMonomer
OEChem::OEMonomerSet, 1036
HasNegativeImageGrid
OEBio::OEReceptor, 1454
HasNucleicAcid
OEBio::OEDesignUnit, 1347
HasOption
OESystem::OEOptions, 1255
HasOriginalBadClusterClash
OEBio::OEPlaceHydrogensMoverNote, 1440
HasOuterContourLevel
OEBio::OEReceptor, 1455
HasPackingResidues
OEBio::OEDesignUnit, 1347
HasParameter
OESystem::OEOptions, 1255
HasPDBMetaData
OEBio::OEDesignUnit, 1347
HasPerceived
OEChem::OEMolBase, 557
HasProperties
OEChem::OEMolPropertyList, 586
HasProtein
OEBio::OEDesignUnit, 1347
HasProteinColorer
OEChem::OE3DMolStyle, 391
HasProteinConstraints
OEBio::OEReceptor, 1455
HasProteinStyle
OEChem::OE3DMolStyle, 391
HasReceptor
OEBio::OEDesignUnit, 1347
HasRGroup
OEChem::OEMonomer, 1028
HasRole
OESystem::OERoleSet, 1163
HasSiteResidues
OEBio::OEDesignUnit, 1347
HasSolvent
OEBio::OEDesignUnit, 1347
HasSpaceGroup
OESystem::OESkewGrid, 1171
HasStereoSpecified
OEChem::OEAtomBase, 405
OEChem::OEBondBase, 425
HasStructureQuality
OEBio::OEDesignUnit, 1347
HasSurfaceAlpha
OEChem::OE3DMolStyle, 391
HasSurfaceColorer
OEChem::OE3DMolStyle, 391
HasSurfaceStyle
OEChem::OE3DMolStyle, 391
HasSurfaceType
OEChem::OE3DMolStyle, 391
HasTarget
OEBio::OEDesignUnit, 1347
HasTitle
OEBio::OEDesignUnit, 1347
OEChem::OEConfBase, 451
HasTransform
OEBio::OEDesignUnit, 1347
HasUnitCell
OESystem::OESkewGrid, 1171
HBonds OEPlaceHydrogens, 218
Helix, 216
Hierarchical View, 214

I

implicit hydrogens, 44
IncrementCount
OESystem::OEErrorHandlerImplBase, 1102
IncrementReadFailureCt
OEChem::oemolistream, 572
Info
OESystem::OEErrorHandler, 1100
Init
OEChem::OECliqueSearch, 446
OEChem::OELibraryGen, 523
OEChem::OELingoSim, 528
OEChem::OEMCSSearch, 547
OEChem::OESubSearch, 627
OEChem::OEUniMolecularRxn, 651
integer atom type, 44
integer bond type, 46
IsAromatic
OEChem::OEAtomBase, 405
OEChem::OEBondBase, 425
IsBitOn
OESystem::OEBitVector, 1079
IsCarbon
OEChem::OEAtomBase, 406
IsCharged
OEBio::OEHBondInteractionHint, 1356
IsChiral
OEChem::OEAtomBase, 406
OEChem::OEBondBase, 425
IsConnected
OEChem::OEAtomBase, 406
IsCTerminal
OEChem::OEMonomer, 1028
IsDataType
OEBio::OEDesignUnit, 1347
OEBio::OEInteractionHint, 1366
OEBio::OEInteractionHintFragment, 1374
OEChem::OEAtomBase, 406
OEChem::OEBondBase, 426
OEChem::OEMolBase, 558

OEChem::OEMolDatabase, 566
OEChem::OEQAtomBase, 605
OEChem::OEQBondBase, 607
OEChem::OEQMolBase, 611
OESystem::OEAnnotation, 1061
OESystem::OEBase, 1065
OESystem::OEBinaryIOHandler, 1073
OESystem::OEContainer, 1095
OESystem::OEFixedGrid, 1108
OESystem::OEFixedMultiGrid, 1109
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
OESystem::OEGaussianContainer, 1115
OESystem::OEGaussianContainerBase, 1117
OESystem::OEGrid, 1119
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1128
OESystem::OEMultiGrid, 1146
OESystem::OEScalarGrid, 1166
OESystem::OESkewGrid, 1171
IsDeleted
OEChem::OEAtomBase, 407
OEChem::OEBondBase, 426
OEChem::OEConfBase, 451
OEChem::OEGroupBase, 473
OEChem::OEMCMolBase, 537
OEChem::OEMolBase, 558
OEChem::OEMolDatabase, 566
IsDirectory
OEPlatform::OEFile, 1266
IsEmpty
OEChem::OEAtomBondSet, 415
OEChem::OEGroupBase, 474
OESystem::OEBitVector, 1079
OESystem::OEColorGradientBase, 1091
IsEquivalent
OEChem::OEExprBase, 465
IsExecutable
OEPlatform::OEFile, 1266
IsFile
OEPlatform::OEFile, 1266
IsFlippable
OEBio::OEPlaceHydrogensMoverNote, 1441
IsFlipped
OEBio::OEPlaceHydrogensMoverNote, 1441
IsHalogen
OEChem::OEAtomBase, 407
IsHetAtom
OEChem::OEResidue, 618
IsHydrogen
OEChem::OEAtomBase, 407
IsInGrid
OESystem::OEGridBase, 1128
IsInRing
OEChem::OEAtomBase, 407
OEChem::OEBondBase, 426
IsInside
OEMath::OEBox, 1303
OEMath::OESphere, 1307
IsInter
OEBio::OEInteractionHint, 1366
IsIntra
OEBio::OEInteractionHint, 1366
IsIonizable
OEBio::OEPlaceHydrogensMoverNote, 1441
IsIonized
OEBio::OEPlaceHydrogensMoverNote, 1441
IsLegalString
OESystem::OEParameter, 1156
IsLink
OEPlatform::OEFile, 1266
IsMetal
OEChem::OEAtomBase, 407
IsNitrogen
OEChem::OEAtomBase, 407
IsNTerminal
OEChem::OEMonomer, 1028
isotopic mass, 44
IsOxygen
OEChem::OEAtomBase, 407
IsPhosphorus
OEChem::OEAtomBase, 408
IsPolar
OEChem::OEAtomBase, 408
IsPolarHydrogen
OEChem::OEAtomBase, 408
IsReadable
OEPlatform::OEFile, 1266
IsRegular
OESystem::OESkewGrid, 1171
IsRestrained
OEBio::OEPlaceHydrogensMoverNote, 1441
IsRotor
OEChem::OEBondBase, 426
IsRxn
OEChem::OEMolBase, 558
IsSet
OESystem::OEParameter, 1156
IsSetToString
OESystem::OEParameter, 1156
IsSingleton
OEBio::OEPlaceHydrogensDetails, 1437
IsSpacingSet
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1128
OESystem::OESkewGrid, 1171
IsSulfur
OEChem::OEAtomBase, 408

IsTitleSet
    OESystem::OEGridBase, 1128
    OESystem::OESkewGrid, 1172
IsValid
    OEBio::OECationPiInteractionHint, 1338
    OEBio::OEChelatorInteractionHint, 1339
    OEBio::OEClashInteractionHint, 1340
    OEBio::OEContactInteractionHint, 1341
    OEBio::OECovalentInteractionHint, 1342
    OEBio::OEHalogenBondInteractionHint, 1349
    OEBio::OEHBondInteractionHint, 1356
    OEBio::OEInteractionHint, 1366
    OEBio::OEInteractionHintComponentTypeBase, 1367
    OEBio::OEInteractionHintContainer, 1371
    OEBio::OEInteractionHintTypeBase, 1375
    OEBio::OELigandInteractionHintComponent, 1413
    OEBio::OEPlaceHydrogensDetails, 1438
    OEBio::OEProteinInteractionHintComponent, 1449
    OEBio::OEReceptor, 1454
    OEBio::OESaltBridgeInteractionHint, 1463
    OEBio::OEStackingInteractionHint, 1473
    OEChem::OE3DMolStyle, 391
    OEChem::OEConfRMSD, 454
    OEChem::OECSVHeader, 462
    OEChem::OELingoSim, 528
    OEChem::OEMatchBase, 532
    OEChem::OEMonomer, 1029
    OEChem::OEMonomerData, 1031
    OEChem::OEMonomerSet, 1036
    OEChem::OESubSearchDatabase, 632
    OEChem::OESubSearchResult, 636
    OEChem::OESubSearchScreen, 639
IsWritable
    OEPlatform::OEFile, 1266
IsXMidSet
    OESystem::OEGridBase, 1128
IsYMidSet
    OESystem::OEGridBase, 1129
IsZMidSet
    OESystem::OEGridBase, 1129
itf interface definition specification, 204

J

Join
    OEPlatform::OEThread, 1272

K

Kekule form, 67
Kekule SMILES, 63
kekulization, 67

L

Lap
    OESystem::OECycleTimer, 1096
    OESystem::OEStopwatch, 1173
    OESystem::OEWallTimer, 1186
LastBit
    OESystem::OEBitVector, 1080
length
    OEPlatform::oestream, 1290
libgen.cs
    Example Code, 280
Load
    OESystem::OEParameter, 1156
LoadDefault
    OESystem::OEParameter, 1156
LoadValue
    OESystem::OEParameter, 1156

M

MakeAltMol
    OEBio::OEAltLocationFactory, 1330
MakeCurrentAltMol
    OEBio::OEAltLocationFactory, 1331
MakePrimaryAltMol
    OEBio::OEAltLocationFactory, 1331
makereceptor.cs
    Example Code, 190
Match
    OEChem::OECliqueSearch, 447
    OEChem::OEMCSSearch, 548
    OEChem::OESubSearch, 628
maximum atom index, 51
maximum bond index, 51
maximum common substructure search, 89
MCMol
    OEChem::OEMol, 552
mcs3dalign.cs
    Example Code, 291
MCSS, 89
MDL query, 103
MinimizeRMSD
    OEChem::OEConfRMSD, 454
minpath.cs
    Example Code, 326
MMCIF data, 32
MMCIF data perception, 33
molchunk.cs
    Example Code, 274
molcount.cs
    Example Code, 319
Molecule constructors
    OEBio::OEDesignUnit, 1345
molecule dimension, 50
molecule energy, 50

molecule title, 50, 51
molextract.cs
    Example Code, 248
molgrep.cs
    Example Code, 285
molstats.cs
    Example Code, 323
Msg
    OESystem::OEErrorHandler, 1100
    OESystem::OEErrorHandlerImplBase, 1102

N

NegateBits
    OESystem::OEBitVector, 1080
NewAtom
    OEChem::OEMolBase, 558
NewBond
    OEChem::OEMolBase, 558
NewConf
    OEChem::OEMCMolBase, 537
NewGroup
    OEChem::OEMolBase, 559
Next
    OEPlatform::OEDirectoryScan, 1265
NextBit
    OESystem::OEBitVector, 1080
NextFloat
    OESystem::OERandom, 1161
NextInt
    OESystem::OERandom, 1161
Null
    OESystem::OEProtectedBuffer, 1160
NumAtoms
    OEBio::OEInteractionHintFragment, 1374
    OEChem::OEAtomBondSet, 416
    OEChem::OEGroupBase, 474
    OEChem::OEMatch, 530
    OEChem::OEMatchBase, 532
    OEChem::OEMolBase, 560
NumAutomorphs
    OEChem::OEConfRMSD, 454
number of atoms, 51
number of bonds, 51
NumBonds
    OEChem::OEAtomBondSet, 416
    OEChem::OEGroupBase, 474
    OEChem::OEMatch, 530
    OEChem::OEMatchBase, 532
    OEChem::OEMolBase, 560
NumCodes
    OEChem::OEMonomer, 1029
    OEChem::OEMonomerData, 1032
NumCodeSets
    OEChem::OEMonomerSet, 1036
NumConfs
    OEChem::OEMCMolBase, 538
NumFeatures
    OEBio::OEReceptorCustomConstraints, 1458
NumFragments
    OEBio::OEInteractionHintContainer, 1372
NumGaussians
    OESystem::OEGaussianContainer, 1116
    OESystem::OEGaussianContainerBase, 1117
NumGroups
    OEChem::OEMolBase, 560
NumInteractions
    OEBio::OEInteractionHintContainer, 1372
NumMatches
    OEChem::OESubSearchDatabase, 632
    OEChem::OESubSearchResult, 636
NumMolecules
    OEBio::OEInteractionHintContainer, 1372
    OEChem::OESubSearchDatabase, 632
NumMols
    OEChem::OEMolDatabase, 566
NumMonomers
    OEChem::OEMonomerSet, 1036
NumPossibleProducts
    OEChem::OELibraryGen, 524
NumProcessors
    OEChem::OESubSearchDatabase, 632
NumProteinConstraints
    OEBio::OEReceptor, 1453
NumReactants
    OEChem::OELibraryGen, 523
NumRGroups
    OEChem::OEMonomer, 1029
NumRings
    OEChem::OE2DRingDictionary, 382
NumRoles
    OESystem::OERoleSet, 1163
NumScreened
    OEChem::OESubSearchResult, 637
NumSearched
    OEChem::OESubSearchResult, 637
NumSmarts
    OEBio::OEReceptorConstraintFeature, 1457
NumSpheres
    OEBio::OEReceptorConstraintFeature, 1456
NumStops
    OESystem::OEColorGradientBase, 1091
NumTargets
    OEChem::OESubSearchResult, 637
NumTotalMatches
    OEChem::OESubSearchResult, 637

O

OEAddExplicitHydrogens, 116

OEAltGroup, 225
OEAltLocation, 225
OEAltLocationFactory, 225
OEAssignAromaticFlags, 135
OEAtomBase, 43
OEAtomIdxSelected, 488
OEBio::OEAltGroup, 1324
    Constructors, 1325
    GetGroupCount, 1325
    GetGroupID, 1325
    GetLocationCount, 1325
    GetLocations, 1325
    operator bool, 1325
    SetGroupID, 1326
OEBio::OEAltLocation, 1326
    Constructors, 1326
    GetGroupID, 1326
    GetLocationCount, 1327
    GetLocationID, 1327
    operator bool, 1326
    SetLocationID, 1327
OEBio::OEAltLocationFactory, 1327
    Constructors, 1327
    GetAltAtoms, 1328
    GetAtomMapSize, 1328
    GetBondMapSize, 1328
    GetCurrentAtoms, 1328
    GetCurrentLocations, 1329
    GetGroup, 1329
    GetGroupCount, 1329
    GetGroups, 1329
    GetLocation, 1329
    GetLocationCodes, 1330
    GetSourceMol, 1330
    GetState, 1330
    MakeAltMol, 1330
    MakeCurrentAltMol, 1331
    MakePrimaryAltMol, 1331
    SetAlt, 1332
    SetPrimaryAlt, 1332
    SetState, 1332
OEBio::OEAltLocationState, 1332
    Constructors, 1333
    operator bool, 1333
    ToHexString, 1333
OEBio::OEAltLocFactOption, 1474
OEBio::OEAltLocFactOption::None, 1474
OEBio::OEAltLocFactOption::TrustInputBonds, 1474
OEBio::OEAssumption, 1474
OEBio::OEAssumption::BondedResidue, 1475
OEBio::OEAssumption::Default, 1475
OEBio::OEAssumption::None, 1475
OEBio::OEAssumption::PDBOrder, 1475
OEBio::OEAssumption::ResPerceived, 1475
OEBio::OEAtomMatchResidue, 1333
    Constructors, 1334
    CreateCopy, 1334
    operator(), 1334
OEBio::OEAtomMatchResidueID, 1334
    Constructors, 1335
    GetAlternateLocation(), 1336
    GetChainID, 1336
    GetFragmentNumber, 1336
    GetInsertCode, 1336
    GetName, 1335
    GetResidueNumber, 1335
    SetAlternateLocation, 1337
    SetChainID, 1336
    SetFragmentNumber, 1336
    SetInsertCode, 1336
    SetName, 1336
    SetResidueNumber, 1336
OEBio::OEAtomMatchResidueIDToString, 1508
OEBio::OEBuildUnitCell, 1508
OEBio::OECalculateDPI, 1509
OEBio::OECationPiInteractionHint, 1337
    Constructors, 1337
    CreateCopy, 1338
    IsValid, 1338
OEBio::OECationPiInteractionHintType, 1475
OEBio::OECationPiInteractionHintType::IntramolecularLigand, 1476
OEBio::OECationPiInteractionHintType::IntramolecularProtei, 1476
OEBio::OECationPiInteractionHintType::LigandCation, 1475
OEBio::OECationPiInteractionHintType::LigandPi, 1475
OEBio::OECationPiInteractionHintType::ProteinCation, 1476
OEBio::OECationPiInteractionHintType::ProteinPi, 1476
OEBio::OEChelatorInteractionHint, 1338
    Constructors, 1339
    CreateCopy, 1339
    GetChelatorType, 1339
    IsValid, 1339
OEBio::OEChelatorInteractionHintType, 1476
OEBio::OEChelatorInteractionHintType::IntramolecularProtei, 1476
OEBio::OEChelatorInteractionHintType::LigandChelates, 1476
OEBio::OEChelatorInteractionHintType::ProteinChelates, 1476
OEBio::OEClashInteractionHint, 1339
    Constructors, 1340
    CreateCopy, 1340

IsValid, 1340
OEBio::OEClearDesignUnitStyle, 1509
OEBio::OEClearMolComplexSDData, 1510
OEBio::OEClearPlaceHydrogensMetadata, 1510
OEBio::OECombineMolComplexFragments, 1510
OEBio::OEConfigureSplitMolComplexOptions, 1511
OEBio::OEConstructInteractionHintContainer, 1543
OEBio::OEContactInteractionHint, 1340
Constructors, 1341
CreateCopy, 1341
IsValid, 1341
OEBio::OECopyCrystalSymmetry, 1512
OEBio::OECountMolComplexSites, 1512
OEBio::OECovalentInteractionHint, 1342
Constructors, 1342
CreateCopy, 1342
IsValid, 1342
OEBio::OECrystalSymmetryParams, 1343
Constructors, 1343
GetA, 1343
GetAlpha, 1343
GetB, 1343
GetBeta, 1344
GetC, 1344
GetGamma, 1344
GetSpaceGroup, 1344
GetZValue, 1344
SetA, 1344
SetAlpha, 1344
SetB, 1344
SetBeta, 1344
SetC, 1344
SetGamma, 1344
SetSpaceGroup, 1345
SetZValue, 1345
OEBio::OEDeleteInteractionsHintSerializationData, 1543
OEBio::OEDeleteInteractionsHintSerializationIds, 1544
OEBio::OEDesignUnit, 1345
Clear, 1346
ClearComponent, 1348
ClearReceptor, 1348
Constructors, 1345
CreateCopy, 1346
GetAltLoc, 1347
GetComponent, 1346
GetComponentID, 1346
GetComponents, 1346
GetComponentsWithin, 1346
GetComponentTag, 1346
GetDataType, 1346
GetIDs, 1346
GetLigand, 1346
GetMetal, 1346
GetNucleicAcid, 1346
GetPackingResidues, 1346
GetPDBMetaData, 1346
GetProtein, 1346
GetReceptor, 1346
GetSiteResidues, 1346
GetSolvent, 1346
GetStructureQuality, 1346
GetTaggedComponents, 1346
GetTags, 1347
GetTarget, 1347
GetTargetComponentID, 1347
GetTargetComponentTag, 1347
GetTitle, 1347
GetTransform, 1346
HasAltLoc, 1347
HasComponent, 1347
HasLigand, 1347
HasMetal, 1347
HasNucleicAcid, 1347
HasPackingResidues, 1347
HasPDBMetaData, 1347
HasProtein, 1347
HasReceptor, 1347
HasSiteResidues, 1347
HasSolvent, 1347
HasStructureQuality, 1347
HasTarget, 1347
HasTitle, 1347
HasTransform, 1347
IsDataType, 1347
Molecule constructors, 1345
operator bool, 1346
operator=, 1345
SetSiteResidues, 1347
SetTitle, 1347
OEBio::OEDesignUnitComponents, 1477
OEBio::OEDesignUnitComponents::All, 1477
OEBio::OEDesignUnitComponents::Cofactors, 1478
OEBio::OEDesignUnitComponents::CounterIons, 1478
OEBio::OEDesignUnitComponents::Default, 1477
OEBio::OEDesignUnitComponents::Excipients, 1478
OEBio::OEDesignUnitComponents::Ligand, 1478
OEBio::OEDesignUnitComponents::Lipids, 1479
OEBio::OEDesignUnitComponents::ListComponents, 1478
OEBio::OEDesignUnitComponents::MacroMolComponents, 1478

OEBio::OEDesignUnitComponents::Metals, 1479
OEBio::OEDesignUnitComponents::MolComponents, 1478
OEBio::OEDesignUnitComponents::Nucleic, 1479
OEBio::OEDesignUnitComponents::OtherCofactors, 1479
OEBio::OEDesignUnitComponents::OtherLigands, 1479
OEBio::OEDesignUnitComponents::OtherNucleics, 1479
OEBio::OEDesignUnitComponents::OtherProteins, 1479
OEBio::OEDesignUnitComponents::PackingResidues, 1479
OEBio::OEDesignUnitComponents::Polymers, 1479
OEBio::OEDesignUnitComponents::PostTranslationalMod, 1480
OEBio::OEDesignUnitComponents::Protein, 1480
OEBio::OEDesignUnitComponents::Solvent, 1480
OEBio::OEDesignUnitComponents::Sugars, 1480
OEBio::OEDesignUnitComponents::TargetComplex, 1477
OEBio::OEDesignUnitComponents::TargetComplexNoSolvents, 1477
OEBio::OEDesignUnitComponents::Undefined, 1480
OEBio::OEDesignUnitComponents::Unknown, 1480
OEBio::OEDesignUnitFileType, 1480
OEBio::OEDesignUnitFileType::OEDesignUnit, 1480
OEBio::OEDesignUnitFileType::Undefined, 1480
OEBio::OEExpandCrystalSymmetry, 1512
OEBio::OEFilterMolComplexFragments, 1513
OEBio::OEGetActiveSiteInteractionHintTypes, 1514
OEBio::OEGetAlignment, 1514
OEBio::OEGetAlignmentMethodName, 1514
OEBio::OEGetAlignments, 1515
OEBio::OEGetAtomMatchResidueIDs, 1516
OEBio::OEGetCalculatedInteractionGeometries, 1516
OEBio::OEGetChis, 1517
OEBio::OEGetCrystalSymmetry, 1517
OEBio::OEGetDesignUnitComponentID, 1518
OEBio::OEGetDesignUnitComponentName, 1518
OEBio::OEGetDesignUnitComponentNamesFromMask, 1519
OEBio::OEGetInteractions, 1519
OEBio::OEGetIridiumCategoryFromName, 1519
OEBio::OEGetIridiumCategoryName, 1520
OEBio::OEGetMolComplexComponents, 1520
OEBio::OEGetMolComplexFilterCategoryName, 1521
OEBio::OEGetMolComplexFragments, 1521
OEBio::OEGetPhi, 1522
OEBio::OEGetPlaceHydrogensMetadata, 1522
OEBio::OEGetPlaceHydrogensMoverClassName, 1522
OEBio::OEGetProteinTorsionName, 1523
OEBio::OEGetPsi, 1523
OEBio::OEGetRamachandranCategoryName, 1523
OEBio::OEGetRamachandranTypeName, 1523
OEBio::OEGetResidueAtom, 1523
OEBio::OEGetResidueAtoms, 1524
OEBio::OEGetResidueDatabaseCategoryName, 1525
OEBio::OEGetResidues, 1525
OEBio::OEGetRotamers, 1525
OEBio::OEGetSecondaryStructureID, 1525
OEBio::OEGetSecondaryStructureName, 1525
OEBio::OEGetSecondaryStructureStrandID, 1526
OEBio::OEGetSecondaryStructureType, 1526
OEBio::OEGetSimpleAlignment, 1526
OEBio::OEGetSpaceGroup, 1527
OEBio::OEGetSpaceGroupNumber, 1527
OEBio::OEGetTorsion, 1527
OEBio::OEHalogenBondInteractionHint, 1348
  Constructors, 1348
  CreateCopy, 1349
  IsValid, 1349
OEBio::OEHalogenBondInteractionHintType, 1481
OEBio::OEHalogenBondInteractionHintType::IntramolecularLig 1481
OEBio::OEHalogenBondInteractionHintType::IntramolecularPro 1481
OEBio::OEHalogenBondInteractionHintType::LigandElectrophil 1481
OEBio::OEHalogenBondInteractionHintType::LigandNucleophile 1481
OEBio::OEHalogenBondInteractionHintType::ProteinElectrophi 1481
OEBio::OEHalogenBondInteractionHintType::ProteinNucleophil 1481
OEBio::OEHasAlternateLocation, 1349
  Constructors, 1350
  CreateCopy, 1350
  operator(), 1350
OEBio::OEHasBondedResidues, 1527
OEBio::OEHasCrystalSymmetry, 1527
OEBio::OEHasInteractionHint, 1350
  Constructors, 1350
  CreateCopy, 1351
  operator(), 1351
OEBio::OEHasInteractionHintSerializationIds, 1544
OEBio::OEHasInteractionHintType, 1351
  Constructors, 1351
  CreateCopy, 1352
  operator(), 1352

OEBio::OEHasInteractionsHintSerializationData, 1544
OEBio::OEHasPDBAtomIndex, 1352
    Constructors, 1352
    CreateCopy, 1352
    operator(), 1352
OEBio::OEHasPlaceHydrogensMetadata, 1528
OEBio::OEHasResidueInteractionHint, 1353
    Constructors, 1353
    CreateCopy, 1353
    operator(), 1353
OEBio::OEHasSecondaryStructure, 1354
    Constructors, 1354
    CreateCopy, 1354
    operator(), 1354
OEBio::OEHBondInteractionHint, 1354
    Constructors, 1355
    CreateCopy, 1355
    GetHBondType, 1355
    IsCharged, 1356
    IsValid, 1356
OEBio::OEHBondInteractionHintType, 1482
OEBio::OEHBondInteractionHintType::Clash, 1482
OEBio::OEHBondInteractionHintType::ClashAcceptorAcceptor, 1482
OEBio::OEHBondInteractionHintType::ClashDonorDonor, 1482
OEBio::OEHBondInteractionHintType::IntramolecularLigand, 1482
OEBio::OEHBondInteractionHintType::IntramolecularProtein, 1482
OEBio::OEHBondInteractionHintType::LigandAccepts, 1482
OEBio::OEHBondInteractionHintType::LigandDonates, 1483
OEBio::OEHBondInteractionHintType::NonIdealLigandAcceptor, 1483
OEBio::OEHBondInteractionHintType::NonIdealLigandDonor, 1483
OEBio::OEHBondInteractionHintType::NonIdealProteinAcceptor, 1483
OEBio::OEHBondInteractionHintType::NonIdealProteinDonor, 1483
OEBio::OEHBondInteractionHintType::ProteinAccepts, 1483
OEBio::OEHBondInteractionHintType::ProteinDonates, 1483
OEBio::OEHBondInteractionHintType::UnpairedLigandAcceptor, 1484
OEBio::OEHBondInteractionHintType::UnpairedLigandDonor, 1483
OEBio::OEHBondInteractionHintType::UnpairedProteinAcceptor, 1484
OEBio::OEHBondInteractionHintType::UnpairedProteinDonor, 1484
OEBio::OEHierChain, 1356
    Constructors, 1356
    GetAtom, 1356
    GetChainID, 1356
    GetFragments, 1356
    GetOEResidue, 1357
    GetResidue, 1357
OEBio::OEHierFragment, 1357
    Constructors, 1357
    GetAtom, 1357
    GetFragmentNumber, 1357
    GetOEResidue, 1357
    GetResidue, 1358
    GetResidues, 1358
OEBio::OEHierResidue, 1358
    Constructors, 1358
    GetAtom, 1358
    GetAtoms, 1358
    GetOEResidue, 1358
    GetResidueName, 1359
    GetResidueNumber, 1359
OEBio::OEHierView, 1359
    Constructors, 1359
    GetAtom, 1360
    GetAtoms, 1360
    GetChains, 1360
    GetFragments, 1360
    GetResidue, 1360
    GetResidues, 1360
OEBio::OEInteractionHint, 1361
    Constructors, 1361
    CreateCopy, 1362
    GetBgnFragment, 1362
    GetCalculatedGeometries, 1364
    GetCalculatedGeometry, 1363
    GetDataType, 1364
    GetDetails, 1364
    GetEndFragment, 1364
    GetFragment, 1365
    GetInteractionType, 1365
    GetWeight, 1366
    HasCalculatedGeometry, 1362
    IsDataType, 1366
    IsInter, 1366
    IsIntra, 1366
    IsValid, 1366
    SetDetails, 1366
    SetWeight, 1366
OEBio::OEInteractionHintComponentTypeBase, 1366
    Constructors, 1367
    CreateCopy, 1367

GetName, 1367
IsValid, 1367
OEBio::OEInteractionHintContainer, 1367
AddMolecule, 1369
Constructors, 1368
DeleteInteraction, 1369
DeleteInteractions, 1369
GetInteractions, 1369
GetMolecule, 1370
GetMolecules, 1371
GetTitle, 1371
HasInteraction, 1371
IsValid, 1371
NumFragments, 1372
NumInteractions, 1372
NumMolecules, 1372
operator bool, 1369
SetTitle, 1372
OEBio::OEInteractionHintFragment, 1372
AddAtom, 1373
Constructors, 1373
CreateCopy, 1373
GetAtom, 1373
GetAtoms, 1373
GetComponentType, 1373
GetDataType, 1374
HasAtom, 1374
IsDataType, 1374
NumAtoms, 1374
OEBio::OEInteractionHintTypeBase, 1374
Constructors, 1375
CreateCopy, 1375
GetName, 1375
IsValid, 1375
OEBio::OEIridiumCategory, 1484
OEBio::OEIridiumCategory::HT, 1484
OEBio::OEIridiumCategory::MT, 1484
OEBio::OEIridiumCategory::NA, 1484
OEBio::OEIridiumCategory::NT, 1484
OEBio::OEIridiumData, 1376
Clear, 1377
Constructors, 1376
DetermineCategory, 1377
GetAltConfs, 1377
GetAsDensity, 1377
GetCategory, 1377
GetDPI, 1377
GetExcipients, 1378
GetHasMTZ, 1378
GetIrrationalRfree, 1378
GetLigandDensity, 1378
GetPackingResidues, 1378
GetPartOccupancyLigand, 1378
GetPartOccupancyProtein, 1378
GetPossibleCovalent, 1378
GetResolution, 1379
GetRFree, 1379
operator bool, 1377
SetAltConfs, 1379
SetAsDensity, 1379
SetCategory, 1379
SetDPI, 1379
SetExcipients, 1379
SetHasMTZ, 1379
SetIrrationalRfree, 1380
SetLigandDensity, 1380
SetPackingResidues, 1380
SetPartOccupancyLigand, 1380
SetPartOccupancyProtein, 1380
SetPossibleCovalent, 1380
SetResolution, 1380
SetRFree, 1380
OEBio::OEIsAlphaCarbon, 1381
OEBio::OEIsBackboneAtom, 1381
Constructors, 1381
CreateCopy, 1382
operator(), 1381
OEBio::OEIsCationPiInteractionHint, 1382
Constructors, 1382
CreateCopy, 1382
operator(), 1382
OEBio::OEIsChelatorInteractionHint, 1382
Constructors, 1383
CreateCopy, 1383
operator(), 1383
OEBio::OEIsClashingHBondInteractionHint, 1383
Constructors, 1384
CreateCopy, 1384
operator(), 1384
OEBio::OEIsClashInteractionHint, 1384
Constructors, 1384
CreateCopy, 1385
operator(), 1384
OEBio::OEIsContactInteractionHint, 1385
Constructors, 1385
CreateCopy, 1385
operator(), 1385
OEBio::OEIsCovalentInteractionHint, 1385
Constructors, 1386
CreateCopy, 1386
operator(), 1386
OEBio::OEIsCTerminalAtom, 1386
Constructors, 1386
CreateCopy, 1387
operator(), 1387
OEBio::OEIsDisulfideBond, 1387
Constructors, 1387
CreateCopy, 1387

operator(), 1387
OEBio::OEIsHalogenBondInteractionHint, 1387
Constructors, 1388
CreateCopy, 1388
operator(), 1388
OEBio::OEIsHBondInteractionHint, 1388
Constructors, 1389
CreateCopy, 1389
operator(), 1389
OEBio::OEIsHetAtom, 1389
Constructors, 1390
CreateCopy, 1390
operator(), 1390
OEBio::OEIsInterInteractionHint, 1390
Constructors, 1390
CreateCopy, 1391
operator(), 1391
OEBio::OEIsIntermolecularCationPiInteractionHint, 1391
Constructors, 1391
CreateCopy, 1391
operator(), 1391
OEBio::OEIsIntermolecularChelatorInteractionHint, 1392
Constructors, 1392
CreateCopy, 1392
operator(), 1392
OEBio::OEIsIntermolecularHalogenBondInteractionHint, 1392
Constructors, 1393
CreateCopy, 1393
operator(), 1393
OEBio::OEIsIntermolecularHBondInteractionHint, 1393
Constructors, 1394
CreateCopy, 1394
operator(), 1394
OEBio::OEIsIntermolecularSaltBridgeInteractionHint, 1394
Constructors, 1395
CreateCopy, 1395
operator(), 1395
OEBio::OEIsIntermolecularStackingInteractionHint, 1395
Constructors, 1396
CreateCopy, 1396
operator(), 1396
OEBio::OEIsIntraInteractionHint, 1396
Constructors, 1396
CreateCopy, 1397
operator(), 1396
OEBio::OEIsIntramolecularCationPiInteractionHint, 1397
Constructors, 1397
CreateCopy, 1397
operator(), 1397
OEBio::OEIsIntramolecularChelatorInteractionHint, 1398
Constructors, 1398
CreateCopy, 1398
operator(), 1398
OEBio::OEIsIntramolecularHalogenBondInteractionHint, 1398
Constructors, 1399
CreateCopy, 1399
operator(), 1399
OEBio::OEIsIntramolecularHBondInteractionHint, 1399
Constructors, 1400
CreateCopy, 1400
operator(), 1400
OEBio::OEIsIntramolecularSaltBridgeInteractionHint, 1400
Constructors, 1401
CreateCopy, 1401
operator(), 1401
OEBio::OEIsIntramolecularStackingInteractionHint, 1401
Constructors, 1401
CreateCopy, 1402
operator(), 1402
OEBio::OEIsNonIdealHBondInteractionHint, 1402
Constructors, 1402
CreateCopy, 1403
operator(), 1402
OEBio::OEIsNTerminalAtom, 1403
Constructors, 1403
CreateCopy, 1403
operator(), 1403
OEBio::OEIsNucleicAcidBase, 1404
CreateCopy, 1404
operator(), 1404
OEBio::OEIsNucleicAcidPhosphate, 1404
CreateCopy, 1405
operator(), 1405
OEBio::OEIsNucleicAcidSugar, 1405
CreateCopy, 1406
operator(), 1406
OEBio::OEIsReadableDesignUnit, 1528
OEBio::OEIsSaltBridgeInteractionHint, 1406
Constructors, 1406
CreateCopy, 1407
operator(), 1407
OEBio::OEIsStackingInteractionHint, 1407
Constructors, 1407
CreateCopy, 1408
operator(), 1407
OEBio::OEIsStandardAminoAcid, 1408

Constructors, 1408
CreateCopy, 1408
operator(), 1408
OEBio::OEIsStandardNucleicAcid, 1408
Constructors, 1409
CreateCopy, 1409
operator(), 1409
OEBio::OEIsStandardNucleicAcidResidue, 1528
OEBio::OEIsStandardProteinResidue, 1528
OEBio::OEIsTypicalPrimaryLocationCode, 1529
OEBio::OEIsUnpairedInteractionHint, 1409
Constructors, 1410
CreateCopy, 1410
operator(), 1410
OEBio::OEIsUnpairedLigandInteractionHint, 1410
Constructors, 1410
CreateCopy, 1411
operator(), 1411
OEBio::OEIsUnpairedProteinInteractionHint, 1411
Constructors, 1411
CreateCopy, 1411
operator(), 1411
OEBio::OEIsValidActiveSite, 1529
OEBio::OEIsWater, 1411
Constructors, 1412
CreateCopy, 1412
operator(), 1412
OEBio::OEIsWriteableDesignUnit, 1530
OEBio::OELigandInteractionHintComponent, 1412
Constructors, 1413
CreateCopy, 1413
GetMaxHeavyAtom, 1413
IsValid, 1413
SetMaxHeavyAtom, 1413
OEBio::OEMakeAltMolOption, 1485
OEBio::OEMakeAltMolOption::None, 1485
OEBio::OEMakeAltMolOption::RemoveAltCodes, 1485
OEBio::OEMolComplexCategorizer, 1413
AddLigandName, 1414
ClearLigandSubSearch, 1415
Constructors, 1414
CreateCopy, 1415
GetBundleCofactors, 1415
GetLigandSubSearch, 1415
GetMaxLigAtoms, 1415
GetMaxLigResidues, 1416
GetMaxPeptideResidues, 1416
GetMaxProteinAtoms, 1416
GetMinLigAtoms, 1416
GetResidueCategoryData, 1416
operator(), 1414
SetBundleCofactors, 1416
SetLigandSubSearch, 1417
SetMaxLigAtoms, 1417
SetMaxLigResidues, 1417
SetMaxPeptideResidues, 1417
SetMaxProteinAtoms, 1417
SetMinLigAtoms, 1418
SetResidueCategoryData, 1418
UsingExplicitLigandName, 1418
OEBio::OEMolComplexCategorizerBase, 1418
CreateCopy, 1419
GetBundleCofactors, 1419
GetResidueCategoryData, 1419
operator(), 1418
UsingExplicitLigandName, 1419
OEBio::OEMolComplexChainRoleFactory, 1530
OEBio::OEMolComplexData, 1485
OEBio::OEMolComplexData::CovalentLigandSDTag, 1485
OEBio::OEMolComplexData::CovalentSDTag, 1485
OEBio::OEMolComplexData::PartOrderSDTag, 1485
OEBio::OEMolComplexData::SDTagPrefix, 1485
OEBio::OEMolComplexData::SiteNumberSDTag, 1485
OEBio::OEMolComplexData::SurfaceWatersNumberSDTag, 1485
OEBio::OEMolComplexFilter, 1419
Constructors, 1419
CreateCopy, 1420
operator(), 1420
OEBio::OEMolComplexFilterCategory, 1486
OEBio::OEMolComplexFilterCategory::Anything, 1486
OEBio::OEMolComplexFilterCategory::ChainA, 1486
OEBio::OEMolComplexFilterCategory::Cofactor, 1486
OEBio::OEMolComplexFilterCategory::CofactorAndLigand, 1486
OEBio::OEMolComplexFilterCategory::CounterIon, 1486
OEBio::OEMolComplexFilterCategory::Covalent, 1486
OEBio::OEMolComplexFilterCategory::DNA_RNA, 1486
OEBio::OEMolComplexFilterCategory::Ligand, 1487
OEBio::OEMolComplexFilterCategory::Lipid, 1487
OEBio::OEMolComplexFilterCategory::Metal, 1487
OEBio::OEMolComplexFilterCategory::Misc, 1487
OEBio::OEMolComplexFilterCategory::Model1, 1487

OEBio::OEMolComplexFilterCategory::Nothing, 1487
OEBio::OEMolComplexFilterCategory::Nucleotide, 1487
OEBio::OEMolComplexFilterCategory::Peptide, 1487
OEBio::OEMolComplexFilterCategory::ProtComplex, 1487
OEBio::OEMolComplexFilterCategory::Protein, 1488
OEBio::OEMolComplexFilterCategory::Saccharide, 1488
OEBio::OEMolComplexFilterCategory::Site1, 1488
OEBio::OEMolComplexFilterCategory::Solution, 1488
OEBio::OEMolComplexFilterCategory::SurfaceWaters1, 1488
OEBio::OEMolComplexFilterCategory::Underdefined, 1488
OEBio::OEMolComplexFilterCategory::Unknown, 1488
OEBio::OEMolComplexFilterCategory::Water, 1488
OEBio::OEMolComplexFilterFactory, 1530
OEBio::OEMolComplexModelRoleFactory, 1531
OEBio::OEMolComplexOrderRoleFactory, 1531
OEBio::OEMolComplexRoleFactory, 1531
OEBio::OEMolComplexSiteRoleFactory, 1532
OEBio::OEMolComplexSurfaceWatersRoleFactory, 1532
OEBio::OEPerceiveInteractionHints, 1532
OEBio::OEPerceiveInteractionOptions, 1420
Constructors, 1421
GetExtendedSaltBridgeDistance, 1421
GetMaxAcceptorAngle, 1422
GetMaxAtomPairDistance, 1422
GetMaxCationPiAngle, 1422
GetMaxCationPiDistance, 1422
GetMaxChargeAidedHBondDistance, 1422
GetMaxContactFraction, 1422
GetMaxCovalentFraction, 1422
GetMaxDonorAngle, 1422
GetMaxHalogenAngle, 1423
GetMaxHalogenBondDistance, 1423
GetMaxHBondDistance, 1423
GetMaxMetalBondDistance, 1423
GetMaxNonIdealAcceptorAngle, 1423
GetMaxNonIdealDonorAngle, 1423
GetMaxNonIdealHBondDistance, 1423
GetMaxPiStackAngle, 1423
GetMaxPiStackDistance, 1424
GetMaxProteinIntraDistance, 1424
GetMaxSaltBridgeDistance, 1424
GetMaxTStackAngle, 1424
GetMaxTStackDistance, 1424
GetMaxUnpairedAngle, 1424
GetMaxUnpairedDistance, 1424
GetMinContactFraction, 1424
GetMinCovalentFraction, 1425
GetMinHBondContactFraction, 1425
GetMinStackOffset, 1425
SetMaxAcceptorAngle, 1425
SetMaxCationPiAngle, 1425
SetMaxCationPiDistance, 1426
SetMaxChargeAidedHBondDistance, 1426
SetMaxContactFraction, 1426
SetMaxCovalentFraction, 1427
SetMaxDonorAngle, 1427
SetMaxHalogenAngle, 1428
SetMaxHalogenBondDistance, 1428
SetMaxHBondDistance, 1428
SetMaxMetalBondDistance, 1429
SetMaxNonIdealAcceptorAngle, 1429
SetMaxNonIdealDonorAngle, 1429
SetMaxNonIdealHBondDistance, 1429
SetMaxPiStackAngle, 1429
SetMaxPiStackDistance, 1430
SetMaxProteinIntraDistance, 1430
SetMaxSaltBridgeDistance, 1430
SetMaxTStackAngle, 1430
SetMaxTStackDistance, 1431
SetMaxUnpairedAngle, 1431
SetMaxUnpairedDistance, 1431
SetMinContactFraction, 1432
SetMinCovalentFraction, 1432
SetMinHBondContactFraction, 1433
SetMinStackOffset, 1433
OEBio::OEPerceiveSecondaryStructure, 1533
OEBio::OEPlaceHydrogens, 1533
OEBio::OEPlaceHydrogensDetails, 1433
ClusterHasBadClash, 1434
Constructors, 1434
Describe, 1434
GetClusterScore, 1436
GetFlipBias, 1436
GetMoverNotes, 1436
GetNumClusters, 1436
GetNumFlips, 1437
GetNumHydrogensStandardized, 1437
GetNumMovers, 1437
IsSingleton, 1437
IsValid, 1438
OEBio::OEPlaceHydrogensMoverClass, 1488
OEBio::OEPlaceHydrogensMoverClass::AmideNO, 1489
OEBio::OEPlaceHydrogensMoverClass::AroCH3, 1489

OEBio::OEPlaceHydrogensMoverClass::AroN, 1489
OEBio::OEPlaceHydrogensMoverClass::Background, 1489
OEBio::OEPlaceHydrogensMoverClass::COOH, 1489
OEBio::OEPlaceHydrogensMoverClass::DH, 1489
OEBio::OEPlaceHydrogensMoverClass::Imidazole, 1489
OEBio::OEPlaceHydrogensMoverClass::Max, 1489
OEBio::OEPlaceHydrogensMoverClass::N, 1489
OEBio::OEPlaceHydrogensMoverClass::NH2, 1490
OEBio::OEPlaceHydrogensMoverClass::NH3, 1490
OEBio::OEPlaceHydrogensMoverClass::NHsp2Only, 1489
OEBio::OEPlaceHydrogensMoverClass::NHsp2sp3, 1490
OEBio::OEPlaceHydrogensMoverClass::NHsp3Only, 1490
OEBio::OEPlaceHydrogensMoverClass::SCH3, 1490
OEBio::OEPlaceHydrogensMoverClass::Water, 1490
OEBio::OEPlaceHydrogensMoverNote, 1438
Constructors, 1438
Describe, 1438
GetAngle, 1438
GetFlippedBestClusterScore, 1438
GetLabel, 1439
GetMoverClass, 1439
GetOriginalBestClusterScore, 1439
GetReferenceCoords, 1439
GetScore, 1439
GetStateIdx, 1440
GetStateName, 1440
HasBadClash, 1440
HasFlippedBadClusterClash, 1440
HasOriginalBadClusterClash, 1440
IsFlippable, 1441
IsFlipped, 1441
IsIonizable, 1441
IsIonized, 1441
IsRestrained, 1441
OEBio::OEPlaceHydrogensOptions, 1442
Constructors, 1442
GetAddMetadata, 1442
GetAltsMustBeCompatible, 1443
GetBadClashOverlapDistance, 1443
GetBypassPredicate, 1443
GetFlipBias, 1443
GetFlipBiasScale, 1444
GetFlipPredicate, 1444
GetMaxSubstateCutoff, 1444
GetNoFlipPredicate, 1444
GetStandardizeBondLen, 1445
GetWaterProcessing, 1445
SetAddMetadata, 1445
SetAltsMustBeCompatible, 1445
SetBadClashOverlapDistance, 1446
SetBypassPredicate, 1446
SetFlipBiasScale, 1446
SetFlipPredicate, 1447
SetMaxSubstateCutoff, 1447
SetNoFlipPredicate, 1447
SetStandardizeBondLen, 1448
SetWaterProcessing, 1448
OEBio::OEPlaceHydrogensWaterProcessing, 1490
OEBio::OEPlaceHydrogensWaterProcessing::Default, 1490
OEBio::OEPlaceHydrogensWaterProcessing::Focused, 1491
OEBio::OEPlaceHydrogensWaterProcessing::FullSearch, 1491
OEBio::OEPlaceHydrogensWaterProcessing::Ignore, 1491
OEBio::OEProteinConstraintType, 1491
OEBio::OEProteinConstraintType::Acceptor, 1491
OEBio::OEProteinConstraintType::Chelator, 1491
OEBio::OEProteinConstraintType::Contact, 1491
OEBio::OEProteinConstraintType::Donor, 1491
OEBio::OEProteinConstraintType::Lipophilic, 1491
OEBio::OEProteinConstraintType::Unknown, 1492
OEBio::OEProteinInteractionHintComponent, 1448
Constructors, 1448
CreateCopy, 1448
IsValid, 1449
OEBio::OEProtTorType, 1492
OEBio::OEProtTorType::Chi1, 1492
OEBio::OEProtTorType::Chi2, 1492
OEBio::OEProtTorType::Chi3, 1492
OEBio::OEProtTorType::Chi4, 1492
OEBio::OEProtTorType::Chi5, 1492
OEBio::OEProtTorType::Max, 1493
OEBio::OEProtTorType::Omega, 1492
OEBio::OEProtTorType::Phi, 1492
OEBio::OEProtTorType::Psi, 1492
OEBio::OEProtTorType::Unknown, 1492
OEBio::OERamaCategory, 1493
OEBio::OERamaCategory::Allowed, 1493
OEBio::OERamaCategory::Favored, 1493
OEBio::OERamaCategory::Outlier, 1493
OEBio::OERamaCategory::Unknown, 1493
OEBio::OERamachandranAnalysis, 1449
Constructors, 1450
GetPhi, 1450
GetPsi, 1450
GetRamaCategory, 1451

GetRamaScore, 1451
GetRamaType, 1451
OEBio::OERamaType, 1493
OEBio::OERamaType::CisPro, 1494
OEBio::OERamaType::General, 1494
OEBio::OERamaType::Glycine, 1494
OEBio::OERamaType::IleVal, 1494
OEBio::OERamaType::PrePro, 1494
OEBio::OERamaType::TransPro, 1494
OEBio::OERamaType::Unknown, 1494
OEBio::OEReadDesignUnit, 1534
OEBio::OEReceptor, 1451
AddProteinConstraint, 1452
Clear, 1452
ClearCustomConstraints, 1452
ClearProteinConstraint, 1452
ClearProteinConstraints, 1452
Constructor, 1452
GetCustomConstraints, 1453
GetInnerContourLevel, 1453
GetInnerContourVolume, 1453
GetNegativeImageGrid, 1453
GetOuterContourLevel, 1453
GetOuterContourVolume, 1453
GetProteinConstraints, 1453
GetTargetMask, 1454
GetTargetPred, 1454
HasCustomConstraints, 1454
HasInnerContourLevel, 1454
HasNegativeImageGrid, 1454
HasOuterContourLevel, 1455
HasProteinConstraints, 1455
IsValid, 1454
NumProteinConstraints, 1453
SetCustomConstraints, 1455
SetInnerContourLevel, 1455
SetOuterContourLevel, 1455
OEBio::OEReceptorConstraintFeature, 1455
AddNewSphere, 1456
AddSmarts, 1456
ClearSmarts, 1457
ClearSpheres, 1456
Constructor, 1455
CreateCopy, 1457
DeleteSmarts, 1457
DeleteSphere, 1456
GetEnabled, 1457
GetFeatureName, 1457
GetFeatureSpheres, 1456
GetSmarts, 1456
NumSmarts, 1457
NumSpheres, 1456
SetEnabled, 1457
SetFeatureName, 1457
OEBio::OEReceptorCustomConstraints, 1458
AddNewFeature, 1458
Clear, 1458
Constructors, 1458
DeleteFeature, 1458
GetConstraintFeatures, 1458
NumFeatures, 1458
OEBio::OEReceptorProteinConstraint, 1459
GetAtom, 1459
GetEnabled, 1459
GetName, 1459
GetType, 1459
SetAtom, 1459
SetEnabled, 1460
SetName, 1459
SetType, 1459
OEBio::OEResidueCategoryData, 1460
AddToDB, 1460
AllInDB, 1461
ClearDBCategory, 1460
Constructors, 1460
FoundInDB, 1461
GetEntries, 1461
RemoveFromDB, 1460
Size, 1461
OEBio::OEResidueDatabaseCategory, 1494
OEBio::OEResidueDatabaseCategory::AminoAcid, 1494
OEBio::OEResidueDatabaseCategory::Cofactor, 1495
OEBio::OEResidueDatabaseCategory::CofactorAndLigand, 1495
OEBio::OEResidueDatabaseCategory::Ligand, 1495
OEBio::OEResidueDatabaseCategory::Lipid, 1495
OEBio::OEResidueDatabaseCategory::Max, 1495
OEBio::OEResidueDatabaseCategory::Metal, 1495
OEBio::OEResidueDatabaseCategory::Misc, 1495
OEBio::OEResidueDatabaseCategory::NucleicAcid, 1495
OEBio::OEResidueDatabaseCategory::Polymer, 1495
OEBio::OEResidueDatabaseCategory::Saccharide, 1495
OEBio::OERMSD, 1534
OEBio::OERoleMolComplexFilterFactory, 1535
OEBio::OERotamer, 1461
GetChi1, 1462
GetChi2, 1462
GetChi3, 1462
GetChi4, 1462
GetProbability, 1462
GetResidueIndex, 1462
OEBio::OERotamerLibrary, 1496

OEBio::OERotamerLibrary::Default, 1496
OEBio::OERotamerLibrary::Dunbrack, 1496
OEBio::OERotamerLibrary::Richardson, 1496
OEBio::OERotamerLibrary::Richardson_2016, 1496
OEBio::OESaltBridgeInteractionHint, 1462
Constructors, 1463
CreateCopy, 1463
IsValid, 1463
OEBio::OESaltBridgeInteractionHintType, 1496
OEBio::OESaltBridgeInteractionHintType::IntramolecularAcidic, 1498
OEBio::OESaltBridgeInteractionHintType::IntramolecularBasic, 1498
OEBio::OESaltBridgeInteractionHintType::LigandNegative, 1496
OEBio::OESaltBridgeInteractionHintType::LigandPositive, 1497
OEBio::OESaltBridgeInteractionHintType::ProteinNegative, 1497
OEBio::OESaltBridgeInteractionHintType::ProteinPositive, 1497
OEBio::OESaltBridgeInteractionHintType::UnpairedLigandAcidic, 1497
OEBio::OESaltBridgeInteractionHintType::UnpairedLigandBasic, 1497
OEBio::OESaltBridgeInteractionHintType::UnpairedLigandNegative, 1497
OEBio::OESaltBridgeInteractionHintType::UnpairedLigandPositive, 1497
OEBio::OESaltBridgeInteractionHintType::UnpairedProteinAcidic, 1497
OEBio::OESaltBridgeInteractionHintType::UnpairedProteinBasic, 1498
OEBio::OESaltBridgeInteractionHintType::UnpairedProteinNegative, 1497
OEBio::OESaltBridgeInteractionHintType::UnpairedProteinPositive, 1497
OEBio::OESecondaryStructure, 1498
OEBio::OESecondaryStructure::Any, 1499
OEBio::OESecondaryStructure::Helix, 1498
OEBio::OESecondaryStructure::Helix310, 1498
OEBio::OESecondaryStructure::HelixAlpha, 1498
OEBio::OESecondaryStructure::HelixLeft, 1499
OEBio::OESecondaryStructure::HelixPi, 1498
OEBio::OESecondaryStructure::Sheet, 1499
OEBio::OESecondaryStructure::StrandSense, 1499
OEBio::OESecondaryStructure::Turn, 1498
OEBio::OESecondaryStructure::Turn3, 1499
OEBio::OESecondaryStructure::Turn4, 1499
OEBio::OESecondaryStructure::Turn5, 1499
OEBio::OESecondaryStructure::Unassigned, 1498
OEBio::OESecondaryStructurePacked, 1535
OEBio::OESeqAlignmentMethod, 1499
OEBio::OESeqAlignmentMethod::BLOSUM62, 1500
OEBio::OESeqAlignmentMethod::GONNET, 1500
OEBio::OESeqAlignmentMethod::Identity, 1499
OEBio::OESeqAlignmentMethod::None, 1499
OEBio::OESeqAlignmentMethod::PAM250, 1500
OEBio::OESequenceAlignment, 1464
Clear, 1464
Constructors, 1464
GetBackboneMatch, 1464
GetCAlphaMatch, 1464
GetChainID, 1464
GetExtend, 1465
GetGap, 1465
GetMaxSeqIdx, 1465
GetMethod, 1465
GetScore, 1465
GetSequenceIdentity, 1465
GetSequenceSimilarity, 1465
operator bool, 1464
OEBio::OESerializeInteractionsHintContainer, 1545
OEBio::OESetCrystalSymmetry, 1535
OEBio::OESetDefaultStyle, 1536
OEBio::OESetInteractionsHintSerializationIds, 1545
OEBio::OESetRotamer, 1537
OEBio::OESetTorsion, 1537
OEBio::OESetupSplitMolComplexOptions, 1537
OEBio::OESplitMolComplex, 1538
OEBio::OESplitMolComplexOptions, 1466
Constructors, 1466
GetAddRGroup, 1466
GetAdjustHCount, 1466
GetAssumptions, 1466
GetCategorizer, 1466
GetCollectProteinByChain, 1467
GetCombinedFilter, 1467
GetLigandFilter, 1467
GetMaxBindingSiteDist, 1467
GetMaxSurfaceWaterDist, 1467
GetOtherFilter, 1467
GetPromoteUnknowns, 1468
GetProteinFilter, 1468
GetSeparateResidues, 1468
GetSplitCovalent, 1468
GetSplitCovalentCofactors, 1468
GetWarnNoLigand, 1468
GetWaterFilter, 1468
ResetFilters, 1468
SetAddRGroup, 1469
SetAdjustHCount, 1469
SetAssumptions, 1469
SetCategorizer, 1469
SetCollectProteinByChain, 1470
SetLigandFilter, 1470
SetMaxBindingSiteDist, 1470
SetMaxSurfaceWaterDist, 1470

SetPromoteUnknowns, 1470
SetProteinFilter, 1470
SetSeparateResidues, 1471
SetSplitCovalent, 1471
SetSplitCovalentCofactors, 1471
SetWarnNoLigand, 1471
SetWaterFilter, 1471
OEBio::OESplitMolComplexParam, 1500
OEBio::OESplitMolComplexParam::DoNotOverride, 1500
OEBio::OESplitMolComplexSetup, 1500
OEBio::OESplitMolComplexSetup::All, 1500
OEBio::OESplitMolComplexSetup::CovBondTreatment, 1501
OEBio::OESplitMolComplexSetup::CovCofactor, 1501
OEBio::OESplitMolComplexSetup::CovLig, 1502
OEBio::OESplitMolComplexSetup::LigFilter, 1502
OEBio::OESplitMolComplexSetup::LigName, 1503
OEBio::OESplitMolComplexSetup::MaxSiteDist, 1503
OEBio::OESplitMolComplexSetup::MaxSurfDist, 1504
OEBio::OESplitMolComplexSetup::ModelNum, 1504
OEBio::OESplitMolComplexSetup::ProtFilter, 1505
OEBio::OESplitMolComplexSetup::SeparateRes, 1505
OEBio::OESplitMolComplexSetup::SiteNum, 1506
OEBio::OESplitMolComplexSetup::SurfWat, 1506
OEBio::OESplitMolComplexSetup::WatFilter, 1507
OEBio::OEStackingInteractionHint, 1472
Constructors, 1472
CreateCopy, 1473
IsValid, 1473
OEBio::OEStackingInteractionHintType, 1507
OEBio::OEStackingInteractionHintType::IntramolecularAromaticRing, 1508
OEBio::OEStackingInteractionHintType::IntramolecularLigandProtein, 1508
OEBio::OEStackingInteractionHintType::IntramolecularProteinAlpha, 1508
OEBio::OEStackingInteractionHintType::IntramolecularProteinSurface, 1508
OEBio::OEStackingInteractionHintType::Pi, 1507
OEBio::OEStackingInteractionHintType::T, 1508
OEBio::OEStructureQuality, 1473
Clear, 1474
Constructors, 1473
GetIridiumData, 1474
HasIridiumData, 1474
operator bool, 1473
SetIridiumData, 1474
OEBio::OESubsetDesignUnit, 1539
OEBio::OESwapAIEResidueAtoms, 1540
OEBio::OESymmetryCartesianToFractional, 1540
OEBio::OESymmetryFractionalToCartesian, 1540
OEBio::OETransformDesignUnit, 1541
OEBio::OEUpdateDesignUnit, 1541
OEBio::OEWriteAlignment, 1540
OEBio::OEWriteDesignUnit, 1542
OEBondBase, 46
OEBondIdxSelected, 488
OEBuildMDLQueryExpression, 105
OEBuildMDLQueryExpressions, 133
OEChem::OE2DRingDictionary, 381
AddRings, 382
Constructors, 381
NumRings, 382
OEChem::OE2DRingDictionaryCreatorOptions, 382
Constructors, 382
GetAddCarbonSkeleton, 383
GetAddCarbonSkeletonWithBondStereo, 383
GetRetainExistingBuiltInTemplates, 384
GetRetainExistingTemplates, 384
SetAddCarbonSkeleton, 384
SetAddCarbonSkeletonWithBondStereo, 384
SetRetainExistingBuiltInTemplates, 385
SetRetainExistingTemplates, 385
OEChem::OE3DMolStyle, 385
ClearAtomColorer, 385
ClearAtomLabelColorer, 386
ClearAtomLabelDataTag, 386
ClearAtomLabelText, 386
ClearAtomLabelType, 386
ClearAtomStyle, 386
ClearContourStyle, 386
ClearGridContourColorer, 386
ClearGridType, 386
ClearHydrogenVisibility, 386
ClearInteractionStyle, 387
ClearLigandColorer, 387
ClearProteinStyle, 387
ClearSurfaceColorer, 387
ClearSurfaceStyle, 387
ClearSurfaceType, 387
Constructors, 385
FromString, 387
GetAtomColorer, 387
GetAtomLabelColorer, 388
GetAtomLabelDataTag, 388
GetAtomLabelPrecision, 388
GetAtomLabelText, 388
GetAtomLabelType, 388

GetAtomStyle, 388
GetContourStyle, 388
GetGridContourColorer, 388
GetGridType, 388
GetHydrogenVisibility, 389
GetInteractionStyle, 389
GetProteinColorer, 389
GetProteinStyle, 389
GetString, 389
GetSurfaceAlpha, 389
GetSurfaceColorer, 389
GetSurfaceStyle, 389
GetSurfaceType, 389
HasAtomColorer, 390
HasAtomLabelColorer, 390
HasAtomLabelDataTag, 390
HasAtomLabelText, 390
HasAtomLabelType, 390
HasAtomStyle, 390
HasContourStyle, 390
HasGridContourColorer, 390
HasGridType, 390
HasHydrogenVisibility, 391
HasInteractionStyle, 391
HasProteinColorer, 391
HasProteinStyle, 391
HasSurfaceAlpha, 391
HasSurfaceColorer, 391
HasSurfaceStyle, 391
HasSurfaceType, 391
IsValid, 391
SetAtomColorer, 392
SetAtomLabelColorer, 392
SetAtomLabelDataTag, 392
SetAtomLabelText, 392
SetAtomLabelType, 392
SetAtomStyle, 392
SetContourStyle, 393
SetGridContourColorer, 393
SetGridType, 393
SetHydrogenVisibility, 393
SetInteractionStyle, 394
SetProteinColorer, 394
SetProteinStyle, 394
SetSurfaceAlpha, 394
SetSurfaceColorer, 394
SetSurfaceStyle, 395
SetSurfaceType, 395
OEChem::OE3DToAtomStereo, 878
OEChem::OE3DToBondStereo, 878
OEChem::OE3DToInternalStereo, 878
OEChem::OEAbsCanonicalConfTest, 396
CombineMols, 396
CompareMols, 396
CreateCopy, 396
OEChem::OEAbsoluteConfTest, 396
CombineMols, 397
CompareMols, 397
Constructors, 397
CreateCopy, 397
OEChem::OEAcyclicPath, 879
OEChem::OEAddCustomFASTAResidue, 879
OEChem::OEAddExplicitHydrogens, 879
OEChem::OEAddMMCIFData, 880
OEChem::OEAddMols, 880
OEChem::OEAddMolStyles, 881
OEChem::OEAddPDBData, 881
OEChem::OEAddSDData, 882
OEChem::OEAminoAcidType, 659
OEChem::OEAminoAcidType::Alpha, 659
OEChem::OEAminoAcidType::AlphaMethylated, 660
OEChem::OEAminoAcidType::Beta, 660
OEChem::OEAminoAcidType::BetaMethylated, 660
OEChem::OEAminoAcidType::Delta, 660
OEChem::OEAminoAcidType::Gamma, 660
OEChem::OEAminoAcidType::NMethylated, 660
OEChem::OEAminoAcidType::NotAminoAcid, 660
OEChem::OEAminoAcidType::Undefined, 660
OEChem::OEAminoAcidTypeToString, 1041
OEChem::OEAnisoUij, 397
Constructors, 398
GetU11, 398
GetU12, 398
GetU13, 398
GetU22, 398
GetU23, 398
GetU33, 398
SetU11, 398
SetU12, 398
SetU13, 398
SetU22, 399
SetU23, 399
SetU33, 399
OEChem::OEApplyStateFromRef, 1042
OEChem::OEApplyStyleChange, 882
OEChem::OEAroModel, 672
OEChem::OEAroModel::Daylight, 672
OEChem::OEAroModel::MDL, 673
OEChem::OEAroModel::MMFF, 673
OEChem::OEAroModel::OpenEye, 672
OEChem::OEAroModel::Tripos, 672
OEChem::OEAssignAromaticFlags, 882
OEChem::OEAssignBondiVdWRadii, 883
OEChem::OEAssignCovalentRadii, 883
OEChem::OEAssignDelphiRadii, 884
OEChem::OEAssignFormalCharges, 884
OEChem::OEAssignHonigIonicCavityRadii, 884
OEChem::OEAssignHybridization, 884

OEChem::OEAssignImplicitHydrogens, 885
OEChem::OEAssignMDLHydrogens, 885
OEChem::OEAssignPaulingVdWRadii, 886
OEChem::OEAssignRadii, 886
OEChem::OEAssignResidueNumbers, 887
OEChem::OEAssignSerialNumbers, 887
OEChem::OEAssignZap7Radii, 887
OEChem::OEAssignZap9Radii, 888
OEChem::OEAtomBase, 399
AddBond, 399
Clear, 399
GetAtomicNum, 399
GetAtoms, 400
GetAxial, 400
GetBond, 400
GetBonds, 400
GetDegree, 400
GetExplicitDegree, 401
GetExplicitHCount, 401
GetExplicitValence, 401
GetFormalCharge, 401
GetHvyDegree, 401
GetHvyValence, 401
GetHyb, 401
GetIdx, 402
GetImplicitHCount, 402
GetIntType, 402
GetIsotope, 402
GetMapIdx, 402
GetName, 403
GetParent, 403
GetPartialCharge, 403
GetRadius, 403
GetRxnRole, 403
GetStereo, 404
GetSymmetryClass, 404
GetTotalHCount, 404
GetType, 404
GetValence, 405
HasAlphaBetaUnsat, 405
HasStereoSpecified, 405
IsAromatic, 405
IsCarbon, 406
IsChiral, 406
IsConnected, 406
IsDataType, 406
IsDeleted, 407
IsHalogen, 407
IsHydrogen, 407
IsInRing, 407
IsMetal, 407
IsNitrogen, 407
IsOxygen, 407
IsPhosphorus, 408
IsPolar, 408
IsPolarHydrogen, 408
IsSulfur, 408
OrderBonds, 408
RemoveBond, 408
SetAromatic, 408
SetAtomicNum, 409
SetAxial, 409
SetChiral, 409
SetFormalCharge, 409
SetHyb, 409
SetImplicitHCount, 410
SetInRing, 410
SetIntType, 410
SetIsotope, 410
SetMapIdx, 410
SetName, 411
SetPartialCharge, 411
SetRadius, 411
SetRxnRole, 411
SetStereo, 411
SetSymmetryClass, 412
SetType, 412
Sweep, 412
OEChem::OEAtomBondSet, 413
Add, 413
AddAtom, 414
AddAtoms, 414
AddBond, 414
AddBonds, 414
Clear, 414
ClearAtoms, 415
ClearAtomsAndBonds, 415
ClearBonds, 415
Constructors, 413
CreateCopy, 415
GetAtoms, 415
GetBonds, 416
GetParent, 416
HasAtom, 415
HasBond, 415
IsEmpty, 415
NumAtoms, 416
NumBonds, 416
Remove, 416
RemoveAtom, 417
RemoveAtoms, 417
RemoveBond, 417
RemoveBonds, 417
Sweep, 417
Translate, 417
OEChem::OEAtomColorScheme, 673
OEChem::OEAtomColorScheme::AltLocation, 673
OEChem::OEAtomColorScheme::Amino, 673

OEChem::OEAtomColorScheme::BEGIN, 673
OEChem::OEAtomColorScheme::BFactor, 674
OEChem::OEAtomColorScheme::Chain, 674
OEChem::OEAtomColorScheme::Element, 674
OEChem::OEAtomColorScheme::END, 674
OEChem::OEAtomColorScheme::FormalCharge, 675
OEChem::OEAtomColorScheme::GenericData, 675
OEChem::OEAtomColorScheme::Occupancy, 675
OEChem::OEAtomColorScheme::PartialCharge, 676
OEChem::OEAtomColorScheme::ProteinSequence, 676
OEChem::OEAtomColorScheme::Residue, 677
OEChem::OEAtomColorScheme::ResidueHydrophobicity, 678
OEChem::OEAtomColorScheme::SecondaryStructure, 679
OEChem::OEAtomColorScheme::Shapely, 680
OEChem::OEAtomColorScheme::Solid, 680
OEChem::OEAtomGetMDLParity, 888
OEChem::OEAtomGetResidue, 888
OEChem::OEAtomGetSmallestRingSize, 888
OEChem::OEAtomIsInAgent, 418
  CreateCopy, 418
  operator(), 418
OEChem::OEAtomIsInAromaticRingSize, 889
OEChem::OEAtomIsInChain, 418
  CreateCopy, 419
  operator(), 419
OEChem::OEAtomIsInProduct, 419
  CreateCopy, 419
  operator(), 419
OEChem::OEAtomIsInReactant, 419
  CreateCopy, 420
  operator(), 420
OEChem::OEAtomIsInResidue, 420
  Constructors, 420
  CreateCopy, 421
  operator(), 421
OEChem::OEAtomIsInRing, 421
  CreateCopy, 421
  operator(), 421
OEChem::OEAtomIsInRingSize, 889
OEChem::OEAtomLabelScheme, 681
OEChem::OEAtomLabelScheme::AltLocation, 681
OEChem::OEAtomLabelScheme::BEGIN, 681
OEChem::OEAtomLabelScheme::BFactor, 681
OEChem::OEAtomLabelScheme::Chirality, 682
OEChem::OEAtomLabelScheme::Degree, 682
OEChem::OEAtomLabelScheme::Element, 683
OEChem::OEAtomLabelScheme::END, 683
OEChem::OEAtomLabelScheme::FormalCharge, 684
OEChem::OEAtomLabelScheme::GenericData, 684
OEChem::OEAtomLabelScheme::HasStereoSpecified, 685
OEChem::OEAtomLabelScheme::Hybridization, 685
OEChem::OEAtomLabelScheme::ImplicitHCount, 685
OEChem::OEAtomLabelScheme::Index, 685
OEChem::OEAtomLabelScheme::IntegerType, 686
OEChem::OEAtomLabelScheme::IsChiral, 686
OEChem::OEAtomLabelScheme::IsInRing, 687
OEChem::OEAtomLabelScheme::Isotope, 687
OEChem::OEAtomLabelScheme::MapIdx, 688
OEChem::OEAtomLabelScheme::MMFFType, 688
OEChem::OEAtomLabelScheme::Occupancy, 688
OEChem::OEAtomLabelScheme::OEChemIdx, 688
OEChem::OEAtomLabelScheme::Off, 688
OEChem::OEAtomLabelScheme::PartialCharge, 689
OEChem::OEAtomLabelScheme::Radius, 689
OEChem::OEAtomLabelScheme::ResidueInfo, 690
OEChem::OEAtomLabelScheme::ResidueInfoCA, 690
OEChem::OEAtomLabelScheme::Symmetry, 691
OEChem::OEAtomLabelScheme::Type, 691
OEChem::OEAtomLabelScheme::User, 691
OEChem::OEAtomSetMDLParity, 890
OEChem::OEAtomSetResidue, 890
OEChem::OEAtomStereo, 691
OEChem::OEAtomStereo::All, 692
OEChem::OEAtomStereo::Left, 692
OEChem::OEAtomStereo::LeftHanded, 692
OEChem::OEAtomStereo::Right, 692
OEChem::OEAtomStereo::RightHanded, 692
OEChem::OEAtomStereo::Tetra, 692
OEChem::OEAtomStereo::Tetrahedral, 692
OEChem::OEAtomStereo::Undefined, 692
OEChem::OEAtomStyle, 693
OEChem::OEAtomStyle::BallAndStick, 693
OEChem::OEAtomStyle::BEGIN, 693
OEChem::OEAtomStyle::CPK, 693
OEChem::OEAtomStyle::END, 694
OEChem::OEAtomStyle::Hidden, 694
OEChem::OEAtomStyle::Stars, 695
OEChem::OEAtomStyle::Stick, 695
OEChem::OEAtomStyle::Wireframe, 696
OEChem::OEBaseColorScheme, 696
OEChem::OEBaseColorScheme::BEGIN, 697
OEChem::OEBaseColorScheme::END, 697
OEChem::OEBaseColorScheme::GenericData, 697
OEChem::OEBaseColorScheme::Solid, 697
OEChem::OEBAtomIntTypeOpts, 890
OEChem::OEBDefaultOpts, 890
OEChem::OEBGZipCompressOpts, 890
OEChem::OEBondBase, 422
  Clear, 422
  GetBgn, 422
  GetBgnIdx, 422
  GetEnd, 423
  GetEndIdx, 423

GetIdx, 423
GetIntType, 423
GetNbr, 423
GetOrder, 424
GetParent, 424
GetStereo, 424
GetType, 424
HasStereoSpecified, 425
IsAromatic, 425
IsChiral, 425
IsDataType, 426
IsDeleted, 426
IsInRing, 426
IsRotor, 426
SetAromatic, 427
SetBgn, 427
SetChiral, 427
SetEnd, 427
SetInRing, 428
SetIntType, 428
SetOrder, 428
SetStereo, 428
SetType, 429
SwapEnds, 429
Sweep, 429
OEChem::OEBondGetSmallestRingSize, 890
OEChem::OEBondIsInAromaticRingSize, 891
OEChem::OEBondIsInChain, 429
CreateCopy, 430
operator(), 430
OEChem::OEBondIsInRing, 430
CreateCopy, 430
operator(), 430
OEChem::OEBondIsInRingSize, 891
OEChem::OEBondStereo, 697
OEChem::OEBondStereo::All, 697
OEChem::OEBondStereo::Cis, 697
OEChem::OEBondStereo::CisTrans, 697
OEChem::OEBondStereo::DoubleEither, 697
OEChem::OEBondStereo::Hash, 698
OEChem::OEBondStereo::Trans, 698
OEChem::OEBondStereo::Undefined, 698
OEChem::OEBondStereo::Wavy, 698
OEChem::OEBondStereo::Wedge, 698
OEChem::OEBRotCompressOpts, 892
OEChem::OEBuildMDLQueryExpressions, 892
OEChem::OECalcCartesianCoord, 892
OEChem::OECalcInternalCoord, 892
OEChem::OECalcPMI, 892
OEChem::OECalculateMolecularWeight, 894
OEChem::OECanonicalOrderAtoms, 894
OEChem::OECanonicalOrderBonds, 895
OEChem::OECenter, 895
OEChem::OECheckAtomValence, 895
OEChem::OECheckAtomValences, 896
OEChem::OEChem, 658
OEChem::OEChem::FzFalse, 659
OEChem::OEChem::FzMaybe, 659
OEChem::OEChem::FzTrue, 659
OEChem::OEChem::OEAroModelDaylight, 658
OEChem::OEChem::OEAroModelMDL, 659
OEChem::OEChem::OEAroModelMMFF, 658
OEChem::OEChem::OEAroModelOpenEye, 658
OEChem::OEChem::OEAroModelTripos, 658
OEChem::OEChemGetArch, 896
OEChem::OEChemGetLicensee, 896
OEChem::OEChemGetPlatform, 896
OEChem::OEChemGetRelease, 896
OEChem::OEChemGetSite, 897
OEChem::OEChemGetVersion, 897
OEChem::OEChemIsLicensed, 897
OEChem::OECIFCategory, 431
AddAttribute, 431
AddRow, 431
Constructors, 431
DeleteAttribute, 432
DeleteRow, 432
GetAttributeIndex, 432
GetAttributes, 432
GetAttributeValue, 432
GetAttributeValues, 432
GetCategoryName, 432
GetDataItem, 433
GetDataItems, 433
GetNumAttributes, 433
GetNumericAttributeValue, 433
GetNumericAttributeValues, 433
GetNumRows, 433
GetRow, 433
GetRowIndices, 434
GetRows, 433
HasAttribute, 434
SetData, 434
SortByAttribute, 434
OEChem::OECIFData, 434
AddCategory, 435
AddData, 436
AddRow, 435
ChemCompToMol, 436
Constructors, 435
DeleteAttribute, 436
DeleteCategory, 436
DeleteRow, 436
GenerateHeader, 436
GeneratePolySeqScheme, 437
GetAttributeIndex, 437
GetAttributes, 437
GetAttributeValue, 437

GetAttributeValues, 437
GetCategory, 438
GetCategoryNames, 438
GetDataItems, 438
GetName, 438
GetNumCategories, 438
GetNumCategoryAttributes, 438
GetNumCategoryRows, 438
GetNumericAttributeValue, 438
GetNumericAttributeValues, 438
GetRow, 439
GetRowIndices, 439
GetRows, 439
HasAttribute, 439
HasCategoryName, 439
SetCategory, 439
SetData, 440
SetMMCIFChemCompData, 440
Update, 440
ValidateMMCIFHeader, 441
ValidateMMCIFMol, 441
OEChem::OECIFOptions, 442
Constructors, 442
GetPerceiveCell, 442
GetPerceiveChemComp, 442
GetPerceiveEntity, 442
GetPerceiveExperimental, 443
GetPerceiveOther, 443
GetPerceiveStruct, 442
GetPerceiveSymmetry, 442
SetPerceiveCell, 443
SetPerceiveChemComp, 443
SetPerceiveEntity, 443
SetPerceiveExperimental, 443
SetPerceiveOther, 444
SetPerceiveStruct, 443
SetPerceiveSymmetry, 443
OEChem::OECIPAtomStereo, 698
OEChem::OECIPAtomStereo::NotStereo, 699
OEChem::OECIPAtomStereo::R, 699
OEChem::OECIPAtomStereo::S, 699
OEChem::OECIPAtomStereo::UnspecStereo, 699
OEChem::OECIPBondStereo, 699
OEChem::OECIPBondStereo::E, 699
OEChem::OECIPBondStereo::NotStereo, 699
OEChem::OECIPBondStereo::UnspecStereo, 700
OEChem::OECIPBondStereo::Z, 699
OEChem::OEClear2DRingDictionary, 897
OEChem::OEClearAromaticFlags, 898
OEChem::OEClearChiralPerception, 898
OEChem::OEClearFormalCharges, 898
OEChem::OEClearMMCIFData, 898
OEChem::OEClearPartialCharges, 899
OEChem::OEClearPDBData, 899
OEChem::OEClearResidue, 899
OEChem::OEClearResidues, 899
OEChem::OEClearRotorCompressData, 899
OEChem::OEClearSDData, 899
OEChem::OEClearStyle, 900
OEChem::OECliqueSearch, 444
AddConstraint, 445
ClearConstraints, 445
Constructors, 444
GetMaxMatches, 445
GetMinAtoms, 445
GetPattern, 446
GetSaveRange, 446
Init, 446
Match, 447
operator bool, 445
SetMaxMatches, 447
SetMCSFunc, 447
SetMinAtoms, 447
SetSaveRange, 448
OEChem::OEConfBase, 448
AddAtom, 449
AddBond, 449
ClearCoords, 449
Delete, 449
GetCoords, 450
GetIdx, 450
GetMCMol, 450
GetTitle, 450
GetTorsion, 451
GetTransform, 451
HasTitle, 451
IsDeleted, 451
RemoveAtom, 452
RemoveBond, 452
SetCoords, 452
SetTorsion, 453
Transform, 453
OEChem::OEConfRMSD, 453
CalculateRMSD, 453
Constructors, 453
GetOptions, 454
IsValid, 454
MinimizeRMSD, 454
NumAutomorphs, 454
OEChem::OEConfRMSDOptions, 455
Constructors, 456
GetIgnoreHydrogens, 456
GetIgnoreTerminalHeavy, 456
GetMaxMatches, 456
SetIgnoreHydrogens, 456
SetIgnoreTerminalHeavy, 456
SetMaxMatches, 456
OEChem::OEConfTestBase, 456

CombineMols, 457
CompareMols, 457
CreateCopy, 457
HasCompareMols, 457
OEChem::OEConstCoords, 457
Constructors, 458
GetPtr, 458
OEChem::OEContourStyle, 700
OEChem::OEContourStyle::BEGIN, 700
OEChem::OEContourStyle::Cloud, 700
OEChem::OEContourStyle::END, 700
OEChem::OEContourStyle::Hidden, 700
OEChem::OEContourStyle::Mesh, 700
OEChem::OEContourStyle::Solid, 701
OEChem::OECoordArray, 458
Constructors, 458
OEChem::OECoordsType, 702
OEChem::OECoordsType::DoubleType, 702
OEChem::OECoordsType::FloatType, 702
OEChem::OECoordsType::HalfFloatType, 702
OEChem::OECoordsType::LongDoubleType, 702
OEChem::OECoordsType::MaxType, 702
OEChem::OECoordsType::Undefined, 702
OEChem::OECopyMMCIFData, 900
OEChem::OECopyMol, 900
OEChem::OECopyPDBData, 901
OEChem::OECopySDData, 901
OEChem::OECorrectAcidProtonGeometry, 902
OEChem::OECount, 902
OEChem::OECreateAbsSmiString, 903
OEChem::OECreateAtomSmartsString, 1043
OEChem::OECreateBondSmartsString, 1043
OEChem::OECreateCanSmiString, 903
OEChem::OECreateCSVHeader, 904
OEChem::OECreateInChI, 904
OEChem::OECreateInChIKey, 904
OEChem::OECreateIsoSmiString, 905
OEChem::OECreateMolDatabaseIdx, 906
OEChem::OECreatePeptideMonomerConnectionSmiles, 1043
OEChem::OECreateSlnString, 907
OEChem::OECreateSmartsString, 1044
OEChem::OECreateSmiString, 907
OEChem::OECreateSubSearchDatabaseFile, 908
OEChem::OECreateSubSearchDatabaseOptions, 458
Constructors, 459
GetKeepTitle, 459
GetNumProcessors, 459
GetScreenType, 460
GetSortByBitCounts, 460
SetKeepTitle, 460
SetNumProcessors, 460
SetSortByBitCounts, 461
OEChem::OECSVHeader, 461
Clear, 461
Constructors, 461
GetColumnName, 462
GetColumnNames, 462
GetNumColumns, 462
IsValid, 462
operator bool, 461
OEChem::OECXSMILESToMol, 1000
OEChem::OEDefaultConfTest, 462
CombineMols, 462
CompareMols, 462
CreateCopy, 463
HasCompareMols, 463
OEChem::OEDefaultImplicitHCount, 910
OEChem::OEDefaultMDLHCount, 910
OEChem::OEDeleteEverythingExceptTheFirstLargestComponent, 910
OEChem::OEDeletePDBData, 911
OEChem::OEDeleteSDData, 911
OEChem::OEDetectMonomers, 1045
OEChem::OEDetermineAromaticRingSystems, 911
OEChem::OEDetermineComponents, 912
OEChem::OEDetermineConnectivity, 912
OEChem::OEDetermineReactants, 913
OEChem::OEDetermineRingSystems, 913
OEChem::OEDisassembleExpressions, 913
OEChem::OEDoubleBondCount, 913
OEChem::OEDoubleBondOCount, 913
OEChem::OEElementHistogram, 914
OEChem::OEElemNo, 702
OEChem::OEElemNo::Ac, 711
OEChem::OEElemNo::Ag, 707
OEChem::OEElemNo::Al, 703
OEChem::OEElemNo::Am, 711
OEChem::OEElemNo::Ar, 704
OEChem::OEElemNo::As, 705
OEChem::OEElemNo::At, 710
OEChem::OEElemNo::Au, 710
OEChem::OEElemNo::B, 702
OEChem::OEElemNo::Ba, 708
OEChem::OEElemNo::Be, 702
OEChem::OEElemNo::Bh, 713
OEChem::OEElemNo::Bi, 710
OEChem::OEElemNo::Bk, 712
OEChem::OEElemNo::Br, 705
OEChem::OEElemNo::C, 703
OEChem::OEElemNo::Ca, 704
OEChem::OEElemNo::Cd, 707
OEChem::OEElemNo::Ce, 708
OEChem::OEElemNo::Cf, 712
OEChem::OEElemNo::Cl, 704
OEChem::OEElemNo::Cm, 712
OEChem::OEElemNo::Cn, 713
OEChem::OEElemNo::Co, 705

OEChem::OEElemNo::Cr, 704
OEChem::OEElemNo::Cs, 707
OEChem::OEElemNo::Cu, 705
OEChem::OEElemNo::D, 714
OEChem::OEElemNo::Db, 712
OEChem::OEElemNo::Ds, 713
OEChem::OEElemNo::Du, 714
OEChem::OEElemNo::Dy, 709
OEChem::OEElemNo::Er, 709
OEChem::OEElemNo::Es, 712
OEChem::OEElemNo::Eu, 708
OEChem::OEElemNo::F, 703
OEChem::OEElemNo::Fe, 705
OEChem::OEElemNo::Fl, 713
OEChem::OEElemNo::Fm, 712
OEChem::OEElemNo::Fr, 711
OEChem::OEElemNo::Ga, 705
OEChem::OEElemNo::Gd, 708
OEChem::OEElemNo::Ge, 705
OEChem::OEElemNo::H, 702
OEChem::OEElemNo::He, 702
OEChem::OEElemNo::Hf, 709
OEChem::OEElemNo::Hg, 710
OEChem::OEElemNo::Ho, 709
OEChem::OEElemNo::Hs, 713
OEChem::OEElemNo::I, 707
OEChem::OEElemNo::In, 707
OEChem::OEElemNo::Ir, 710
OEChem::OEElemNo::K, 704
OEChem::OEElemNo::Kr, 706
OEChem::OEElemNo::La, 708
OEChem::OEElemNo::Li, 702
OEChem::OEElemNo::Lp, 714
OEChem::OEElemNo::Lr, 712
OEChem::OEElemNo::Lu, 709
OEChem::OEElemNo::Lv, 714
OEChem::OEElemNo::MAXELEM, 714
OEChem::OEElemNo::Mc, 713
OEChem::OEElemNo::Md, 712
OEChem::OEElemNo::Mg, 703
OEChem::OEElemNo::Mn, 704
OEChem::OEElemNo::Mo, 706
OEChem::OEElemNo::Mt, 713
OEChem::OEElemNo::N, 703
OEChem::OEElemNo::Na, 703
OEChem::OEElemNo::Nb, 706
OEChem::OEElemNo::Nd, 708
OEChem::OEElemNo::Ne, 703
OEChem::OEElemNo::Nh, 713
OEChem::OEElemNo::Ni, 705
OEChem::OEElemNo::No, 712
OEChem::OEElemNo::Np, 711
OEChem::OEElemNo::O, 703
OEChem::OEElemNo::Og, 714
OEChem::OEElemNo::Os, 710
OEChem::OEElemNo::P, 703
OEChem::OEElemNo::Pa, 711
OEChem::OEElemNo::Pb, 710
OEChem::OEElemNo::Pd, 707
OEChem::OEElemNo::Pm, 708
OEChem::OEElemNo::Po, 710
OEChem::OEElemNo::Pr, 708
OEChem::OEElemNo::Pt, 710
OEChem::OEElemNo::Pu, 711
OEChem::OEElemNo::Ra, 711
OEChem::OEElemNo::Rb, 706
OEChem::OEElemNo::Re, 709
OEChem::OEElemNo::Rf, 712
OEChem::OEElemNo::Rg, 713
OEChem::OEElemNo::Rh, 706
OEChem::OEElemNo::Rn, 711
OEChem::OEElemNo::Ru, 706
OEChem::OEElemNo::S, 704
OEChem::OEElemNo::Sb, 707
OEChem::OEElemNo::Sc, 704
OEChem::OEElemNo::Se, 705
OEChem::OEElemNo::Sg, 713
OEChem::OEElemNo::Si, 703
OEChem::OEElemNo::Sm, 708
OEChem::OEElemNo::Sn, 707
OEChem::OEElemNo::Sr, 706
OEChem::OEElemNo::T, 714
OEChem::OEElemNo::Ta, 709
OEChem::OEElemNo::Tb, 708
OEChem::OEElemNo::Tc, 706
OEChem::OEElemNo::Te, 707
OEChem::OEElemNo::Th, 711
OEChem::OEElemNo::Ti, 704
OEChem::OEElemNo::Tl, 710
OEChem::OEElemNo::Tm, 709
OEChem::OEElemNo::Ts, 714
OEChem::OEElemNo::U, 711
OEChem::OEElemNo::V, 704
OEChem::OEElemNo::W, 709
OEChem::OEElemNo::Xe, 707
OEChem::OEElemNo::Xx, 714
OEChem::OEElemNo::Y, 706
OEChem::OEElemNo::Yb, 709
OEChem::OEElemNo::Zn, 705
OEChem::OEElemNo::Zr, 706
OEChem::OEEuler, 463
    Constructors, 463
    CreateCopy, 463
    Extract, 463
    GetAngles, 463
    GetInverse, 464
    GetPhi, 464
    GetPsi, 464

GetTheta, 464
SetPhi, 464
SetPsi, 464
SetTheta, 464
OEChem::OEEulerRotate, 914
OEChem::OEExactGraphMatch, 914
OEChem::OEExpandSuperAtoms, 914
OEChem::OEExprBase, 464
Constructors, 464
CreateCopy, 465
GetType, 465
IsEquivalent, 465
OEChem::OEExprOpts, 714
OEChem::OEExprOpts::Aromaticity, 716
OEChem::OEExprOpts::AtomicNumber, 716
OEChem::OEExprOpts::AutomorphAtoms, 717
OEChem::OEExprOpts::AutomorphBonds, 717
OEChem::OEExprOpts::BondOrder, 717
OEChem::OEExprOpts::Chiral, 719
OEChem::OEExprOpts::DefaultAtoms, 718
OEChem::OEExprOpts::DefaultBonds, 718
OEChem::OEExprOpts::Degree, 720
OEChem::OEExprOpts::EqAromatic, 720
OEChem::OEExprOpts::EqCAliphaticONS, 721
OEChem::OEExprOpts::EqCHalogen, 721
OEChem::OEExprOpts::EqCPSAcidRoot, 722
OEChem::OEExprOpts::EqDoubleTriple, 722
OEChem::OEExprOpts::EqHalogen, 723
OEChem::OEExprOpts::EqKetoneSulfoneRoot, 723
OEChem::OEExprOpts::EqMetal, 724
OEChem::OEExprOpts::EqNotAromatic, 724
OEChem::OEExprOpts::EqON, 725
OEChem::OEExprOpts::EqONS, 726
OEChem::OEExprOpts::EqPS, 726
OEChem::OEExprOpts::EqSingleDouble, 726
OEChem::OEExprOpts::ExactAtoms, 727
OEChem::OEExprOpts::ExactBonds, 727
OEChem::OEExprOpts::ExplicitDegree, 728
OEChem::OEExprOpts::FormalCharge, 728
OEChem::OEExprOpts::HCount, 728
OEChem::OEExprOpts::HvyDegree, 729
OEChem::OEExprOpts::Hybridization, 729
OEChem::OEExprOpts::ImplicitHCount, 729
OEChem::OEExprOpts::IntType, 729
OEChem::OEExprOpts::Mass, 729
OEChem::OEExprOpts::RingMember, 729
OEChem::OEExprOpts::StrictFormalCharge, 731
OEChem::OEExprOpts::StringType, 732
OEChem::OEExprOpts::SymmetryClass, 732
OEChem::OEExprOpts::Valence, 732
OEChem::OEExprType, 732
OEChem::OEExprType::AllowedElements, 732
OEChem::OEExprType::And, 732
OEChem::OEExprType::Aromatic, 732
OEChem::OEExprType::AromBondOrder, 733
OEChem::OEExprType::AromElem, 733
OEChem::OEExprType::AromElemMinDegree, 733
OEChem::OEExprType::AromOrBondOrder, 733
OEChem::OEExprType::AromRingElem, 733
OEChem::OEExprType::AtomParityChange, 733
OEChem::OEExprType::BondOrder, 733
OEChem::OEExprType::BondParityChange, 733
OEChem::OEExprType::Charge, 733
OEChem::OEExprType::Chiral, 733
OEChem::OEExprType::Connect, 733
OEChem::OEExprType::Const, 733
OEChem::OEExprType::DefaultEdge, 734
OEChem::OEExprType::Degree, 734
OEChem::OEExprType::DoubleAromBond, 734
OEChem::OEExprType::ElemCharge, 734
OEChem::OEExprType::Element, 734
OEChem::OEExprType::ElemMinDegree, 734
OEChem::OEExprType::ElemMinHCount, 734
OEChem::OEExprType::FormalChargeChange, 734
OEChem::OEExprType::Halogen, 734
OEChem::OEExprType::HCount, 734
OEChem::OEExprType::HCountChange, 734
OEChem::OEExprType::Hyb, 734
OEChem::OEExprType::Implicit, 734
OEChem::OEExprType::IntType, 735
OEChem::OEExprType::Mass, 735
OEChem::OEExprType::Metal, 735
OEChem::OEExprType::MinDegree, 735
OEChem::OEExprType::MinHCount, 735
OEChem::OEExprType::Not, 735
OEChem::OEExprType::Or, 735
OEChem::OEExprType::Recurs, 735
OEChem::OEExprType::Ring, 735
OEChem::OEExprType::RingBondCount, 735
OEChem::OEExprType::RingBondOrder, 735
OEChem::OEExprType::RingDefaultEdge, 735
OEChem::OEExprType::RingElem, 735
OEChem::OEExprType::Saturation, 736
OEChem::OEExprType::SingleAromBond, 736
OEChem::OEExprType::Size, 736
OEChem::OEExprType::Stereo, 736
OEChem::OEExprType::StringType, 736
OEChem::OEExprType::SymmetryClass, 736
OEChem::OEExprType::TerminalAtom, 736
OEChem::OEExprType::Undefined, 736
OEChem::OEExprType::Valence, 736
OEChem::OEExtractRingTemplateOptions, 465
Constructors, 466
GetRetainBondStereo, 466
GetRetainExistingTemplates, 466
SetRetainBondStereo, 466
SetRetainExistingTemplates, 467
OEChem::OEExtractRingTemplates, 914

OEChem::OEFastaToMol, 915
OEChem::OEFileStringParameter, 467
    Constructors, 468
OEChem::OEFileStringType, 736
OEChem::OEFileStringType::BroodQMol, 737
OEChem::OEFileStringType::DU, 737
OEChem::OEFileStringType::Mol, 736
OEChem::OEFileStringType::Mol3D, 736
OEChem::OEFileStringType::MolReceptor, 736
OEChem::OEFileStringTypeFlavor, 737
OEChem::OEFileStringTypeFlavor::DEFAULT, 737
OEChem::OEFileStringTypeFlavor::Default, 737
OEChem::OEFileStringTypeFlavor::Dim2D, 737
OEChem::OEFileStringTypeFlavor::Dim2D3D, 737
OEChem::OEFileStringTypeFlavor::Dim3D, 737
OEChem::OEFileStringTypeFlavor::FileList, 737
OEChem::OEFileStringTypeFlavor::Input, 738
OEChem::OEFileStringTypeFlavor::InputDefault, 738
OEChem::OEFileStringTypeFlavor::MolOnly, 738
OEChem::OEFileStringTypeFlavor::NoListDefault, 738
OEChem::OEFileStringTypeFlavor::Output, 738
OEChem::OEFileStringTypeFlavor::OutputDefault, 738
OEChem::OEFileStringTypeFlavor::Pretty, 737
OEChem::OEFileStringTypeFlavor::RxnOnly, 738
OEChem::OEFindRingAtomsAndBonds, 915
OEChem::OEFormalPartialCharges, 915
OEChem::OEFormat, 738
OEChem::OEFormat::CAN, 739
OEChem::OEFormat::CDX, 739
OEChem::OEFormat::CIF, 739
OEChem::OEFormat::CSV, 739
OEChem::OEFormat::CXSMILES, 739
OEChem::OEFormat::Default, 738
OEChem::OEFormat::FASTA, 739
OEChem::OEFormat::INCHI, 739
OEChem::OEFormat::INCHIKEY, 740
OEChem::OEFormat::ISM, 740
OEChem::OEFormat::MAXFORMAT, 740
OEChem::OEFormat::MDL, 740
OEChem::OEFormat::MF, 740
OEChem::OEFormat::MMCIF, 740
OEChem::OEFormat::MMOD, 740
OEChem::OEFormat::MOL2, 740
OEChem::OEFormat::MOL2H, 740
OEChem::OEFormat::MOPAC, 741
OEChem::OEFormat::OEB, 741
OEChem::OEFormat::OEZ, 741
OEChem::OEFormat::PDB, 741
OEChem::OEFormat::RDF, 741
OEChem::OEFormat::SDF, 741
OEChem::OEFormat::SKC, 741
OEChem::OEFormat::SLN, 742
OEChem::OEFormat::SMI, 742
OEChem::OEFormat::UNDEFINED, 742
OEChem::OEFormat::USM, 742
OEChem::OEFormat::XYZ, 742
OEChem::OEFuzzVal, 742
OEChem::OEFuzzVal::False, 742
OEChem::OEFuzzVal::Maybe, 742
OEChem::OEFuzzVal::True, 742
OEChem::OEFuzzy, 469
    Constructors, 469
    operator bool, 469
OEChem::OEGasteigerInitialCharges, 916
OEChem::OEGasteigerPartialCharges, 916
OEChem::OEGenerate2DCoordinates, 916
OEChem::OEGetAbsTorsion, 917
OEChem::OEGetAminoAcidCode, 917
OEChem::OEGetAminoAcidType, 1046
OEChem::OEGetAngle, 918
OEChem::OEGetAnisou, 919
OEChem::OEGetAtomColor, 919
OEChem::OEGetAtomComment, 919
OEChem::OEGetAtomicNum, 919
OEChem::OEGetAtomicSymbol, 920
OEChem::OEGetAtomLabelColor, 919
OEChem::OEGetAtomLabelText, 919
OEChem::OEGetAutomorphs, 920
OEChem::OEGetAverageWeight, 920
OEChem::OEGetBondiVdWRadius, 920
OEChem::OEGetCenterAndExtents, 920
OEChem::OEGetCenterOfMass, 921
OEChem::OEGetColorSchemeName, 921
OEChem::OEGetComment, 921
OEChem::OEGetCovalentRadius, 921
OEChem::OEGetDefaultIFlavor, 922
OEChem::OEGetDefaultMass, 922
OEChem::OEGetDefaultOFlavor, 922
OEChem::OEGetDelphiRadius, 922
OEChem::OEGetDimensionFromCoords, 923
OEChem::OEGetDistance, 923
OEChem::OEGetDistance2, 923
OEChem::OEGetDoubleBondNeighbor, 924
OEChem::OEGetFileExtension, 924
OEChem::OEGetFileType, 924
OEChem::OEGetFormatExtension, 924
OEChem::OEGetFormatString, 925
OEChem::OEGetHelmChainNames, 1046
OEChem::OEGetHelmMonomerCodes, 1046
OEChem::OEGetHonigIonicCavityRadius, 925
OEChem::OEGetHybridization, 925
OEChem::OEGetHydrogenCounts, 925
OEChem::OEGetIsotopicWeight, 926
OEChem::OEGetMCSExhaustiveSearchTruncationLimit, 926

OEChem::OEGetMDLDimensionFromCoords, 926
OEChem::OEGetMMCIFData, 929
OEChem::OEGetMolDatabaseIdxFileName, 927
OEChem::OEGetMolDBIdx, 927
OEChem::OEGetMonomerType, 1047
OEChem::OEGetNbrAtom, 469
    Constructors, 470
    CreateCopy, 470
    operator(), 470
OEChem::OEGetNearestNbrs, 927
OEChem::OEGetNucleicAcidCode, 929
OEChem::OEGetOEGraphMol, 929
OEChem::OEGetOEMol, 930
OEChem::OEGetOEQMol, 930
OEChem::OEGetOtherNeighbor, 930
OEChem::OEGetPackedCoords, 930
OEChem::OEGetPathLength, 931
OEChem::OEGetPaulingVdWRadius, 931
OEChem::OEGetPDBAtomIndex, 931
OEChem::OEGetPDBAtomName, 932
OEChem::OEGetPDBBaseData, 932
OEChem::OEGetPDBData, 932
OEChem::OEGetPDBDataPairs, 932
OEChem::OEGetPolymerType, 1048
OEChem::OEGetRadiiTypeName, 932
OEChem::OEGetRadiusOfGyration, 933
OEChem::OEGetResidueIndex, 934
OEChem::OEGetResidueIndexFromCode, 935
OEChem::OEGetResidueName, 935
OEChem::OEGetSDBaseData, 935
OEChem::OEGetSDData, 936
OEChem::OEGetSDDataPairs, 935
OEChem::OEGetSingleBondNeighbor, 936
OEChem::OEGetSmallestSubtree, 936
OEChem::OEGetSmiStringOrder, 937
OEChem::OEGetSoleBond, 937
OEChem::OEGetSoleDoubleBond, 937
OEChem::OEGetSoleNeighbor, 937
OEChem::OEGetSoleSingleBond, 937
OEChem::OEGetSoleTripleBond, 938
OEChem::OEGetStandardAnalog, 1048
OEChem::OEGetStyle, 938
OEChem::OEGetStyleChange, 938
OEChem::OEGetStyleIndex, 938
OEChem::OEGetSubSearchScreenType, 938
OEChem::OEGetSubtree, 939
OEChem::OEGetTokenizer, 939
OEChem::OEGetTorsion, 940
OEChem::OEGetTorsions, 940
OEChem::OEGetTripleBondNeighbor, 941
OEChem::OEGetUniMolecularRxnIter, 941
OEChem::OEGraphMol, 470
    Constructors, 471
    SCMol, 471
OEChem::OEGroupBase, 471
    AddAtom, 471
    AddBond, 472
    Clear, 472
    DeleteAtom, 472
    DeleteBond, 472
    GetAtoms, 472
    GetBonds, 472
    GetGroupDefinition, 473
    GetGroupType, 473
    GetParent, 473
    HasAtom, 473
    HasBond, 473
    IsDeleted, 473
    IsEmpty, 474
    NumAtoms, 474
    NumBonds, 474
    SetGroupType, 474
    Sweep, 474
OEChem::OEGroupType, 742
OEChem::OEGroupType::Component, 742
OEChem::OEGroupType::MDLAbsStereo, 743
OEChem::OEGroupType::MDLAndStereo, 743
OEChem::OEGroupType::MDLOrStereo, 743
OEChem::OEGroupType::Monomer, 743
OEChem::OEGroupType::Undefined, 743
OEChem::OEGroupType::UserDefined, 743
OEChem::OEHasAlphaBetaUnsat, 474
    Constructors, 475
    CreateCopy, 475
    operator(), 475
OEChem::OEHasAminoAcidType, 1018
    Constructors, 1019
    CreateCopy, 1019
    operator(), 1019
OEChem::OEHasAnisou, 475
    CreateCopy, 475
    operator(), 475
OEChem::OEHasAtomicNum, 475
    Constructors, 476
    CreateCopy, 476
    operator(), 476
OEChem::OEHasAtomInGroup, 476
    Constructors, 476
    CreateCopy, 477
    operator(), 477
OEChem::OEHasAtomName, 477
    Constructors, 477
    CreateCopy, 477
    operator(), 477
OEChem::OEHasAtomStereoHydrogens, 941
OEChem::OEHasAtomStereoSpecified, 478
    Constructors, 478
    CreateCopy, 478

operator(), 478
OEChem::OEHasBondInGroup, 478
Constructors, 479
CreateCopy, 479
operator(), 479
OEChem::OEHasBondStereoHydrogens, 941
OEChem::OEHasBondStereoSpecified, 479
Constructors, 480
CreateCopy, 480
operator(), 480
OEChem::OEHasChainID, 480
Constructors, 480
CreateCopy, 481
operator(), 480
OEChem::OEHasDoubleBond, 942
OEChem::OEHasDoubleBondO, 941
OEChem::OEHasExplicitHydrogenExpression, 942
OEChem::OEHasExplicitHydrogens, 942
OEChem::OEHasFormalCharge, 481
Constructors, 481
CreateCopy, 481
operator(), 481
OEChem::OEHasFragmentNumber, 481
Constructors, 482
CreateCopy, 482
operator(), 482
OEChem::OEHasGroupType, 482
Constructors, 483
CreateCopy, 483
operator(), 483
OEChem::OEHasHvyDegree, 483
Constructors, 483
CreateCopy, 483
operator(), 483
OEChem::OEHasIdx, 484
Constructors, 484
CreateCopy, 484
operator(), 484
OEChem::OEHasImplicitHydrogenExpression, 942
OEChem::OEHasImplicitHydrogens, 942
OEChem::OEHasMapIdx, 485
Constructors, 485
CreateCopy, 485
operator(), 485
OEChem::OEHasMMCIFData, 942
OEChem::OEHasMolDBIdx, 943
OEChem::OEHasMonomerGroup, 1019
Constructors, 1020
CreateCopy, 1020
OEChem::OEHasMonomerType, 1020
Constructors, 1020
CreateCopy, 1020
operator(), 1020
OEChem::OEHasMultipleBond, 943
OEChem::OEHasOrder, 485
Constructors, 486
CreateCopy, 486
operator(), 486
OEChem::OEHasPartialCharges, 943
OEChem::OEHasPDBData, 943
OEChem::OEHasPolymerType, 1020
Constructors, 1021
CreateCopy, 1021
operator(), 1021
OEChem::OEHasRadii, 943
OEChem::OEHasResidue, 944
OEChem::OEHasResidueNumber, 486
Constructors, 486
CreateCopy, 487
operator(), 487
OEChem::OEHasResidues, 944
OEChem::OEHasSDData, 944
OEChem::OEHasSingleBond, 945
OEChem::OEHasStereoHydrogens, 945
OEChem::OEHasStyle, 945
OEChem::OEHasTokenizer, 945
OEChem::OEHasTripleBond, 945
OEChem::OEHelmGenerationOptions, 1021
Constructors, 1021
GetAllowUnmatchedFragments, 1022
GetAllowUnspecifiedStereo, 1022
GetCodeSet, 1022
GetMinUnmatchedAtomsRatio, 1022
SetAllowUnmatchedFragments, 1022
SetAllowUnspecifiedStereo, 1022
SetCodeSet, 1023
SetMinUnmatchedAtomsRatio, 1023
OEChem::OEHelmGenerationResult, 1023
Constructors, 1023
GetCodeSet, 1023
GetReturnCode, 1023
GetUnmappedFragments, 1024
GetWarning, 1024
OEChem::OEHelmGenerationReturnCode, 660
OEChem::OEHelmGenerationReturnCode::InvalidCodeSet, 661
OEChem::OEHelmGenerationReturnCode::InvalidMonomerGraph, 661
OEChem::OEHelmGenerationReturnCode::InvalidUnmappedFragmer 661
OEChem::OEHelmGenerationReturnCode::NoSequence, 661
OEChem::OEHelmGenerationReturnCode::Success, 661
OEChem::OEHelmGenerationReturnCode::TooLarge, 661
OEChem::OEHelmGenerationReturnCode::UnmappedFragments, 661

OEChem::OEHelmGenerationReturnCode::UnspecifiedAtomGenerator, 662
OEChem::OEHelmGenerationReturnCode::UnspecifiedBondStereo, 662
OEChem::OEHelmParsingResult, 1024
    Constructors, 1024
    GetCodeSet, 1024
    GetErrorPosition, 1024
    GetReturnCode, 1024
    GetWarning, 1025
OEChem::OEHelmParsingReturnCode, 662
OEChem::OEHelmParsingReturnCode::HelmInvalidComponent, 663
OEChem::OEHelmParsingReturnCode::HelmInvalidVersion, 663
OEChem::OEHelmParsingReturnCode::HelmSyntaxError, 663
OEChem::OEHelmParsingReturnCode::InvalidConnectionChainId, 663
OEChem::OEHelmParsingReturnCode::InvalidConnectionMonomerIdx, 663
OEChem::OEHelmParsingReturnCode::InvalidConnectionRGroupIdx, 663
OEChem::OEHelmParsingReturnCode::InvalidEmbeddedSmiles, 663
OEChem::OEHelmParsingReturnCode::InvalidMonomerCode, 663
OEChem::OEHelmParsingReturnCode::NotSupported, 663
OEChem::OEHelmParsingReturnCode::Success, 662
OEChem::OEHelmParsingReturnCode::UnrecognizedMonomerCode, 663
OEChem::OEHelmToMol, 1049
OEChem::OEHistogramFormula, 945
OEChem::OEHybridization, 743
OEChem::OEHybridization::sp, 743
OEChem::OEHybridization::sp2, 744
OEChem::OEHybridization::sp3, 744
OEChem::OEHybridization::sp3d, 744
OEChem::OEHybridization::sp3d2, 744
OEChem::OEHybridization::Unknown, 743
OEChem::OEHydrogenCounter, 487
    Constructors, 487
    GetExplicitHCount, 487
    GetImplicitHCount, 487
OEChem::OEHydrogenVisibility, 744
OEChem::OEHydrogenVisibility::All, 744
OEChem::OEHydrogenVisibility::BEGIN, 744
OEChem::OEHydrogenVisibility::END, 745
OEChem::OEHydrogenVisibility::Off, 745
OEChem::OEHydrogenVisibility::Polar, 745
OEChem::OEIdxSelected, 487
    Constructors, 488
    CreateCopy, 488
    operator(), 488
OEChem::OEIFlavor, 746
OEChem::OEIFlavor::CAN, 746
OEChem::OEIFlavor::CAN::Default, 747
OEChem::OEIFlavor::CDX, 747
OEChem::OEIFlavor::CDX::Default, 747
OEChem::OEIFlavor::CDX::SuperAtom, 747
OEChem::OEIFlavor::CIF, 747
OEChem::OEIFlavor::CIF::BondHydToClosest, 748
OEChem::OEIFlavor::CIF::BondOrder, 747
OEChem::OEIFlavor::CIF::Default, 748
OEChem::OEIFlavor::CIF::FormalCrg, 747
OEChem::OEIFlavor::CIF::ImplicitH, 747
OEChem::OEIFlavor::CIF::NormalizeHydPos, 748
OEChem::OEIFlavor::CIF::OccFilterOneHalf, 748
OEChem::OEIFlavor::CIF::RemovePBCImages, 748
OEChem::OEIFlavor::CIF::RemoveQuestionMarkInLabel, 748
OEChem::OEIFlavor::CIF::Rings, 748
OEChem::OEIFlavor::CSV, 748
OEChem::OEIFlavor::CSV::DEFAULT, 749
OEChem::OEIFlavor::CSV::Default, 749
OEChem::OEIFlavor::CSV::Header, 749
OEChem::OEIFlavor::CXSMILES, 749
OEChem::OEIFlavor::CXSMILES::Canon, 749
OEChem::OEIFlavor::CXSMILES::DEFAULT, 749
OEChem::OEIFlavor::CXSMILES::Default, 749
OEChem::OEIFlavor::CXSMILES::Strict, 749
OEChem::OEIFlavor::FASTA, 750
OEChem::OEIFlavor::FASTA::AllMask, 750
OEChem::OEIFlavor::FASTA::CustomResidues, 750
OEChem::OEIFlavor::FASTA::Default, 750
OEChem::OEIFlavor::FASTA::EmbeddedSMILES, 750
OEChem::OEIFlavor::Generic, 750
OEChem::OEIFlavor::Generic::AroMask, 751
OEChem::OEIFlavor::Generic::DEFAULT, 752
OEChem::OEIFlavor::Generic::Default, 752
OEChem::OEIFlavor::Generic::GenericMask, 751
OEChem::OEIFlavor::Generic::OEAroModelDaylight, 750
OEChem::OEIFlavor::Generic::OEAroModelMDL, 751
OEChem::OEIFlavor::Generic::OEAroModelMMFF, 751
OEChem::OEIFlavor::Generic::OEAroModelOpenEye, 751
OEChem::OEIFlavor::Generic::OEAroModelTripos, 751
OEChem::OEIFlavor::Generic::Rings, 751
OEChem::OEIFlavor::Generic::SpecificMask, 752
OEChem::OEIFlavor::ISM, 752
OEChem::OEIFlavor::ISM::Default, 752
OEChem::OEIFlavor::MDL, 752
OEChem::OEIFlavor::MDL::Default, 752

OEChem::OEIFlavor::MDL::FixBondMarks, 752
OEChem::OEIFlavor::MDL::ForceImp2ExpENHSTE, 753
OEChem::OEIFlavor::MDL::SuppressEmptyMolSkip, 752
OEChem::OEIFlavor::MDL::SuppressImp2ExpENHSTE, 753
OEChem::OEIFlavor::MMCIF, 753
OEChem::OEIFlavor::MMCIF::AllMask, 754
OEChem::OEIFlavor::MMCIF::ALTLOC, 754
OEChem::OEIFlavor::MMCIF::BondOrder, 753
OEChem::OEIFlavor::MMCIF::Connect, 753
OEChem::OEIFlavor::MMCIF::DEFAULT, 754
OEChem::OEIFlavor::MMCIF::Default, 754
OEChem::OEIFlavor::MMCIF::FormalCrg, 753
OEChem::OEIFlavor::MMCIF::ImplicitH, 753
OEChem::OEIFlavor::MMCIF::Rings, 753
OEChem::OEIFlavor::MMCIF::SecStruct, 753
OEChem::OEIFlavor::MMCIF::SpruceDefault, 754
OEChem::OEIFlavor::MMOD, 754
OEChem::OEIFlavor::MMOD::Default, 754
OEChem::OEIFlavor::MMOD::FormalCrg, 754
OEChem::OEIFlavor::MOL2, 754
OEChem::OEIFlavor::MOL2::AllMask, 755
OEChem::OEIFlavor::MOL2::DEFAULT, 755
OEChem::OEIFlavor::MOL2::Default, 755
OEChem::OEIFlavor::MOL2::Forcefield, 755
OEChem::OEIFlavor::MOL2::M2H, 755
OEChem::OEIFlavor::MOL2H, 755
OEChem::OEIFlavor::MOL2H::AllMask, 755
OEChem::OEIFlavor::MOL2H::DEFAULT, 756
OEChem::OEIFlavor::MOL2H::Default, 756
OEChem::OEIFlavor::MOL2H::M2H, 755
OEChem::OEIFlavor::OEB, 756
OEChem::OEIFlavor::OEB::Default, 756
OEChem::OEIFlavor::OEZ, 756
OEChem::OEIFlavor::OEZ::Default, 756
OEChem::OEIFlavor::PDB, 756
OEChem::OEIFlavor::PDB::ALL, 757
OEChem::OEIFlavor::PDB::AllMask, 760
OEChem::OEIFlavor::PDB::ALTLOC, 757
OEChem::OEIFlavor::PDB::BasicMask, 758
OEChem::OEIFlavor::PDB::BondOrder, 759
OEChem::OEIFlavor::PDB::CHARGE, 758
OEChem::OEIFlavor::PDB::Connect, 759
OEChem::OEIFlavor::PDB::DATA, 757
OEChem::OEIFlavor::PDB::DEFAULT, 760
OEChem::OEIFlavor::PDB::Default, 760
OEChem::OEIFlavor::PDB::DELPHI, 758
OEChem::OEIFlavor::PDB::END, 756
OEChem::OEIFlavor::PDB::ENDM, 756
OEChem::OEIFlavor::PDB::ExtraMask, 760
OEChem::OEIFlavor::PDB::FORMALCHARGE, 758
OEChem::OEIFlavor::PDB::FormalCrg, 758
OEChem::OEIFlavor::PDB::ImplicitH, 759
OEChem::OEIFlavor::PDB::RADIUS, 758
OEChem::OEIFlavor::PDB::Rings, 759
OEChem::OEIFlavor::PDB::SecStruct, 758
OEChem::OEIFlavor::PDB::SpruceDefault, 760
OEChem::OEIFlavor::PDB::TER, 756
OEChem::OEIFlavor::PDB::TerMask, 756
OEChem::OEIFlavor::RDF, 760
OEChem::OEIFlavor::RDF::Default, 760
OEChem::OEIFlavor::RDF::FixBondMarks, 760
OEChem::OEIFlavor::RDF::ForceImp2ExpENHSTE, 761
OEChem::OEIFlavor::RDF::SuppressEmptyMolSkip, 761
OEChem::OEIFlavor::RDF::SuppressImp2ExpENHSTE, 761
OEChem::OEIFlavor::SDF, 761
OEChem::OEIFlavor::SDF::Default, 761
OEChem::OEIFlavor::SDF::FixBondMarks, 761
OEChem::OEIFlavor::SDF::ForceImp2ExpENHSTE, 761
OEChem::OEIFlavor::SDF::SuppressEmptyMolSkip, 761
OEChem::OEIFlavor::SDF::SuppressImp2ExpENHSTE, 761
OEChem::OEIFlavor::SKC, 762
OEChem::OEIFlavor::SKC::Default, 762
OEChem::OEIFlavor::SMI, 762
OEChem::OEIFlavor::SMI::Canon, 762
OEChem::OEIFlavor::SMI::DEFAULT, 762
OEChem::OEIFlavor::SMI::Default, 762
OEChem::OEIFlavor::SMI::Strict, 762
OEChem::OEIFlavor::USM, 762
OEChem::OEIFlavor::USM::Default, 762
OEChem::OEIFlavor::XYZ, 763
OEChem::OEIFlavor::XYZ::BondOrder, 763
OEChem::OEIFlavor::XYZ::Connect, 763
OEChem::OEIFlavor::XYZ::Default, 763
OEChem::OEIFlavor::XYZ::ExtraMask, 763
OEChem::OEIFlavor::XYZ::FormalCrg, 763
OEChem::OEIFlavor::XYZ::ImplicitH, 763
OEChem::OEIFlavor::XYZ::Rings, 763
OEChem::OEInChIOptions, 489
    Constructors, 489
    GetChiral, 489
    GetFixedHLayer, 489
    GetHydrogens, 489
    GetNonPolymerPseudoAtoms, 489
    GetRacemicStereo, 490
    GetReconnectedMetals, 490
    GetRelativeStereo, 490
    GetStereo, 490
    GetTimeout, 490
    GetUndefinedStereo, 490

SetChiral, 490
SetFixedHLayer, 491
SetHydrogens, 491
SetNonPolymerPseudoAtoms, 491
SetRacemicStereo, 491
SetReconnectedMetals, 491
SetRelativeStereo, 491
SetStereo, 491
SetTimeout, 492
SetUndefinedStereo, 492
OEChem::OEInChIToMol, 945
OEChem::OEInit2DRingDictionary, 946
OEChem::OEInitDefaultHandler, 946
OEChem::OEInitHandler, 946
OEChem::OEInSamePart, 492
Constructors, 492
CreateCopy, 492
operator(), 492
OEChem::OEInteractionStyle, 763
OEChem::OEInteractionStyle::BEGIN, 764
OEChem::OEInteractionStyle::Bumps, 764
OEChem::OEInteractionStyle::END, 764
OEChem::OEInteractionStyle::IntermolecularHBonds, 764
OEChem::OEInteractionStyle::IntramolecularHBonds, 764
OEChem::OEInteractionStyle::Off, 765
OEChem::OEInteractionStyle::UnsatisfiedHBonds, 766
OEChem::OEInvertCenter, 947
OEChem::OEIs2DFormat, 947
OEChem::OEIs3DFormat, 947
OEChem::OEIsAmideBond, 493
Constructors, 493
CreateCopy, 493
operator(), 493
OEChem::OEIsAromaticAtom, 493
CreateCopy, 494
operator(), 494
OEChem::OEIsAromaticBond, 494
CreateCopy, 494
operator(), 494
OEChem::OEIsAtomHybridization, 494
Constructors, 495
CreateCopy, 495
operator(), 495
OEChem::OEIsBinary, 947
OEChem::OEIsBridgeHead, 495
CreateCopy, 496
operator(), 495
OEChem::OEIsCAlpha, 496
CreateCopy, 496
operator(), 496
OEChem::OEIsCarbon, 496
CreateCopy, 497
operator(), 497
OEChem::OEIsChiralAtom, 497
CreateCopy, 498
operator(), 497
OEChem::OEIsChiralBond, 498
CreateCopy, 498
operator(), 498
OEChem::OEIsCIFHeaderCompliant, 947
OEChem::OEIsCommonIsotope, 948
OEChem::OEIsComplementaryScreenType, 948
OEChem::OEIsDisulfideBridgeBond, 498
Constructors, 499
CreateCopy, 499
operator(), 499
OEChem::OEIsExtension, 949
OEChem::OEIsGZip, 949
OEChem::OEIsHalide, 499
CreateCopy, 499
operator(), 499
OEChem::OEIsHalogen, 500
CreateCopy, 500
operator(), 500
OEChem::OEIsHeavy, 500
CreateCopy, 501
operator(), 500
OEChem::OEIsHetero, 501
CreateCopy, 501
operator(), 501
OEChem::OEIsHydrogen, 501
CreateCopy, 502
operator(), 502
OEChem::OEIsInMonomerCodeSet, 1025
Constructors, 1025
CreateCopy, 1025
operator(), 1025
OEChem::OEIsIntraResidueBond, 502
Constructors, 502
CreateCopy, 503
operator(), 502
OEChem::OEIsInvertibleNitrogen, 503
CreateCopy, 503
operator(), 503
OEChem::OEIsMDLStereoGroup, 503
Constructors, 504
CreateCopy, 504
operator(), 504
OEChem::OEIsMember, 504
Add, 505
Constructors, 505
CreateCopy, 505
operator(), 505
Set, 505
OEChem::OEIsMemberPtr, 506

Add, 507
Constructors, 506
CreateCopy, 507
operator(), 506
Set, 507
OEChem::OEIsMetal, 507
CreateCopy, 507
operator(), 507
OEChem::OEIsMonomerGroup, 1025
Constructors, 1026
CreateCopy, 1026
OEChem::OEIsNitrogen, 507
CreateCopy, 508
operator(), 508
OEChem::OEIsNonRingAtomDoubleBondedToRing, 508
CreateCopy, 508
operator(), 508
OEChem::OEIsomericConfTest, 509
CombineMols, 509
CompareMols, 510
Constructors, 509
CreateCopy, 510
OEChem::OEIsOxygen, 510
CreateCopy, 510
operator(), 510
OEChem::OEIsPeptide, 1050
OEChem::OEIsPhosphorus, 510
CreateCopy, 511
operator(), 511
OEChem::OEIsPolar, 511
CreateCopy, 511
operator(), 511
OEChem::OEIsPolarHydrogen, 511
CreateCopy, 512
operator(), 512
OEChem::OEIsProductGroup, 512
Constructors, 512
CreateCopy, 513
operator(), 513
OEChem::OEIsReactantGroup, 513
Constructors, 513
CreateCopy, 514
operator(), 513
OEChem::OEIsReadable, 949
OEChem::OEIsRGroup, 514
Constructors, 514
CreateCopy, 514
operator(), 514
OEChem::OEIsRotor, 515
Constructors, 515
CreateCopy, 515
operator(), 515
OEChem::OEIsRxnComponentGroup, 515
Constructors, 516
CreateCopy, 516
operator(), 516
OEChem::OEIsSDDataFormat, 949
OEChem::OEIsStandardMonomer, 1048
OEChem::OEIsSulfur, 516
CreateCopy, 517
operator(), 517
OEChem::OEIsTermHeavyAtom, 517
CreateCopy, 517
operator(), 517
OEChem::OEIsTerminalOxygen, 949
OEChem::OEIsTerminalSulfur, 949
OEChem::OEIsValid2DRingDictionary, 949
OEChem::OEIsValidAtomValence, 518
CreateCopy, 518
operator(), 518
OEChem::OEIsValidMDLAtomValence, 518
CreateCopy, 519
operator(), 519
OEChem::OEIsValidMonomerData, 1050
OEChem::OEIsValidSubSearchDatabase, 950
OEChem::OEIsValidSubSearchScreenString, 951
OEChem::OEIsWriteable, 951
OEChem::OEKekulize, 951
OEChem::OELibraryGen, 519
AddStartingMaterial, 520
ClearStartingMaterial, 521
Constructors, 519
GetAssignMapIdx, 521
GetClearCoordinates, 521
GetExplicitHydrogens, 522
GetProducts, 522
GetRemoveUnmappedFragments, 522
GetStartingMaterial, 522
GetTitleSeparator, 523
GetValenceCorrection, 523
GetValidateKekule, 523
Init, 523
NumPossibleProducts, 524
NumReactants, 523
operator bool, 520
operator(), 520
SetAssignMapIdx, 524
SetClearCoordinates, 524
SetExplicitHydrogens, 525
SetRemoveUnmappedFragments, 525
SetStartingMaterial, 525
SetTitleSeparator, 526
SetValenceCorrection, 526
SetValidateKekule, 526
OEChem::OELigandEnergy, 671
OEChem::OELingoSim, 527
Constructors, 527

GetType, 527
Init, 528
IsValid, 528
Similarity, 528
OEChem::OELingoType, 766
OEChem::OELingoType::Default, 766
OEChem::OELingoType::IsoSmiMap, 766
OEChem::OELingoType::Undefined, 766
OEChem::OELoadOpenEyeMonomerSet, 1051
OEChem::OELoadStandardMonomerSet, 1052
OEChem::OEMacroModelAtomTypeNames, 952
OEChem::OEMacroModelAtomTypes, 952
OEChem::OEMacroModelTypeElement, 952
OEChem::OEMacroModelTypeName, 952
OEChem::OEMacroModelTypeNames, 952
OEChem::OEMakeSubSearchQueryScreen, 952
OEChem::OEMakeSubSearchTargetScreen, 953
OEChem::OEMatch, 528
AddPair, 529
Clear, 530
Constructors, 529
CreateCopy, 530
GetAtoms, 530
GetBonds, 530
NumAtoms, 530
NumBonds, 530
OEChem::OEMatchBase, 531
Clear, 531
CreateCopy, 531
GetAtoms, 531
GetBonds, 531
GetPatternAtoms, 532
GetPatternBonds, 532
GetTargetAtoms, 532
GetTargetBonds, 532
IsValid, 532
NumAtoms, 532
NumBonds, 532
operator bool, 531
OEChem::OEMatchFunc, 532
Constructors, 533
CreateCopy, 534
operator bool, 534
operator(), 533
OEChem::OEMatchPair, 534
Constructors, 534
OEChem::OEMCMolBase, 534
ClearBase, 535
ClearMCMol, 535
ClearMolBase, 535
DeleteConf, 536
DeleteConfs, 536
GetActive, 536
GetConf, 536
GetConfs, 537
GetMaxConfIdx, 537
GetMCMolTitle, 537
IsDeleted, 537
NewConf, 537
NumConfs, 538
OrderConfs, 538
PopActive, 538
PushActive, 539
SetActive, 539
SweepConfs, 539
OEChem::OEMCMolType, 766
OEChem::OEMCMolType::Cartesian, 767
OEChem::OEMCMolType::DoubleCartesian, 767
OEChem::OEMCMolType::FloatCartesian, 767
OEChem::OEMCMolType::HalfFloatCartesian, 767
OEChem::OEMCMolType::LongDoubleCartesian, 767
OEChem::OEMCMolType::MaxType, 768
OEChem::OEMCMolType::OEDBMCMol, 767
OEChem::OEMCMolType::OEDefault, 767
OEChem::OEMCMolType::Undefined, 768
OEChem::OEMCSFunc, 539
CreateCopy, 540
operator(), 540
OEChem::OEMCSFuncParameter, 541
Constructors, 541
OEChem::OEMCSMaxAtoms, 542
CreateCopy, 542
operator(), 542
OEChem::OEMCSMaxAtomsCompleteCycles, 542
Constructors, 543
CreateCopy, 543
operator(), 543
OEChem::OEMCSMaxBonds, 543
CreateCopy, 544
operator(), 544
OEChem::OEMCSMaxBondsCompleteCycles, 544
Constructors, 545
CreateCopy, 545
operator(), 545
OEChem::OEMCSSearch, 545
AddConstraint, 546
ClearConstraints, 547
Constructors, 546
GetMaxMatches, 547
GetMinAtoms, 547
GetPattern, 547
Init, 547
Match, 548
operator bool, 546
SetMaxMatches, 549
SetMCSFunc, 549
SetMinAtoms, 549
OEChem::OEMCSType, 768

OEChem::OEMCSType::Approximate, 768
OEChem::OEMCSType::Default, 768
OEChem::OEMCSType::Exhaustive, 768
OEChem::OEMDLClearBondStereo, 954
OEChem::OEMDLClearParity, 954
OEChem::OEMDLCorrectBondStereo, 955
OEChem::OEMDLGetParity, 955
OEChem::OEMDLGetValence, 956
OEChem::OEMDLHasIncorrectBondStereo, 957
OEChem::OEMDLHasParity, 957
OEChem::OEMDLPerceiveBondStereo, 958
OEChem::OEMDLPerceiveParity, 958
OEChem::OEMDLQueryOpts, 768
OEChem::OEMDLQueryOpts::AddBondAliphaticConstraint, 769
OEChem::OEMDLQueryOpts::AddBondTopologyConstraint, 769
OEChem::OEMDLQueryOpts::Default, 768
OEChem::OEMDLQueryOpts::ExactMatchStarAtoms, 769
OEChem::OEMDLQueryOpts::MatchAtomStereo, 770
OEChem::OEMDLQueryOpts::MatchIsotope, 770
OEChem::OEMDLQueryOpts::Optimize, 770
OEChem::OEMDLQueryOpts::ReactionQuery, 770
OEChem::OEMDLQueryOpts::SuppressExplicitH, 769
OEChem::OEMDLSetParity, 959
OEChem::OEMDLStereoFromBondStereo, 960
OEChem::OEMDLStereoFromParity, 961
OEChem::OEMMFF94InitialCharges, 962
OEChem::OEMMFF94PartialCharges, 962
OEChem::OEMMFFAtomType, 962
OEChem::OEMMFFAtomTypeNames, 962
OEChem::OEMMFFAtomTypes, 963
OEChem::OEMMFFClearRemappedElements, 964
OEChem::OEMMFFRemapElement, 964
OEChem::OEMMFFType, 770
OEChem::OEMMFFType::B, 786
OEChem::OEMMFFType::BR, 775
OEChem::OEMMFFType::Br, 775
OEChem::OEMMFFType::BR_, 785
OEChem::OEMMFFType::C, 772
OEChem::OEMMFFType::C5, 784
OEChem::OEMMFFType::C5A, 782
OEChem::OEMMFFType::C5B, 782
OEChem::OEMMFFType::C__, 782
OEChem::OEMMFFType::CA2, 785
OEChem::OEMMFFType::CB, 780
OEChem::OEMMFFType::CC, 771
OEChem::OEMMFFType::CGD, 771
OEChem::OEMMFFType::CGD_, 782
OEChem::OEMMFFType::CIM, 784
OEChem::OEMMFFType::CION, 787
OEChem::OEMMFFType::CL, 775
OEChem::OEMMFFType::Cl, 775
OEChem::OEMMFFType::CL_, 785
OEChem::OEMMFFType::CLO4, 784
OEChem::OEMMFFType::CN, 771
OEChem::OEMMFFType::CNN_, 782
OEChem::OEMMFFType::CO, 771
OEChem::OEMMFFType::CO2M, 780
OEChem::OEMMFFType::CON, 771
OEChem::OEMMFFType::CONN, 786
OEChem::OEMMFFType::COO, 771
OEChem::OEMMFFType::COON, 771
OEChem::OEMMFFType::COOO, 771
OEChem::OEMMFFType::COR, 771
OEChem::OEMMFFType::COS, 771
OEChem::OEMMFFType::CP, 772
OEChem::OEMMFFType::CR, 771
OEChem::OEMMFFType::CR3R, 776
OEChem::OEMMFFType::CR4E, 778
OEChem::OEMMFFType::CR4R, 776
OEChem::OEMMFFType::CS, 771
OEChem::OEMMFFType::CS2M, 780
OEChem::OEMMFFType::CSN, 772
OEChem::OEMMFFType::CSO, 772
OEChem::OEMMFFType::CSO2, 772
OEChem::OEMMFFType::CSP, 772
OEChem::OEMMFFType::CSP2, 771
OEChem::OEMMFFType::CSS, 772
OEChem::OEMMFFType::CU1, 785
OEChem::OEMMFFType::CU2, 785
OEChem::OEMMFFType::DUMMY, 772
OEChem::OEMMFFType::F, 774
OEChem::OEMMFFType::F_, 785
OEChem::OEMMFFType::FE2, 784
OEChem::OEMMFFType::FE3, 785
OEChem::OEMMFFType::H3N, 776
OEChem::OEMMFFType::HC, 772
OEChem::OEMMFFType::HGD_, 780
OEChem::OEMMFFType::HIM, 779
OEChem::OEMMFFType::HN5_, 786
OEChem::OEMMFFType::HNC, 777
OEChem::OEMMFFType::HNC_, 777
OEChem::OEMMFFType::HNC__, 780
OEChem::OEMMFFType::HNCC, 777
OEChem::OEMMFFType::HNCN, 777
OEChem::OEMMFFType::HNCO, 777
OEChem::OEMMFFType::HNCS, 777
OEChem::OEMMFFType::HNM, 786
OEChem::OEMMFFType::HNN, 777
OEChem::OEMMFFType::HNN_, 780
OEChem::OEMMFFType::HNNC, 777
OEChem::OEMMFFType::HNNN, 777
OEChem::OEMMFFType::HNO, 786
OEChem::OEMMFFType::HNO2, 786
OEChem::OEMMFFType::HNOX, 776

OEChem::OEMMFFType::HNPO, 786
OEChem::OEMMFFType::HNR, 776
OEChem::OEMMFFType::HNR_, 779
OEChem::OEMMFFType::HNSO, 777
OEChem::OEMMFFType::HO, 776
OEChem::OEMMFFType::HO_, 781
OEChem::OEMMFFType::HO__, 781
OEChem::OEMMFFType::HOCC, 778
OEChem::OEMMFFType::HOCN, 778
OEChem::OEMMFFType::HOCO, 776
OEChem::OEMMFFType::HOCS, 785
OEChem::OEMMFFType::HOH, 778
OEChem::OEMMFFType::HOM, 785
OEChem::OEMMFFType::HON, 786
OEChem::OEMMFFType::HOP, 776
OEChem::OEMMFFType::HOR, 776
OEChem::OEMMFFType::HOS, 779
OEChem::OEMMFFType::HP, 783
OEChem::OEMMFFType::HPD_, 780
OEChem::OEMMFFType::HPYL, 776
OEChem::OEMMFFType::HS, 783
OEChem::OEMMFFType::HSE, 787
OEChem::OEMMFFType::HSe, 787
OEChem::OEMMFFType::HSI, 772
OEChem::OEMMFFType::HSN, 783
OEChem::OEMMFFType::HSP2, 778
OEChem::OEMMFFType::HX, 786
OEChem::OEMMFFType::I, 775
OEChem::OEMMFFType::K, 785
OEChem::OEMMFFType::LI, 785
OEChem::OEMMFFType::MAXTYPE, 787
OEChem::OEMMFFType::MG2, 785
OEChem::OEMMFFType::N2OX, 783
OEChem::OEMMFFType::N3OX, 783
OEChem::OEMMFFType::N5, 784
OEChem::OEMMFFType::N5_, 784
OEChem::OEMMFFType::N5A, 782
OEChem::OEMMFFType::N5A_, 784
OEChem::OEMMFFType::N5AX, 784
OEChem::OEMMFFType::N5B, 783
OEChem::OEMMFFType::N5B_, 784
OEChem::OEMMFFType::N5BX, 784
OEChem::OEMMFFType::N5M, 784
OEChem::OEMMFFType::N5OX, 784
OEChem::OEMMFFType::N_, 781
OEChem::OEMMFFType::NA, 785
OEChem::OEMMFFType::NAZT, 781
OEChem::OEMMFFType::NC, 774
OEChem::OEMMFFType::NC_, 782
OEChem::OEMMFFType::NCC, 780
OEChem::OEMMFFType::NCC_, 780
OEChem::OEMMFFType::NCN, 780
OEChem::OEMMFFType::NCN_, 781
OEChem::OEMMFFType::NCN__, 782
OEChem::OEMMFFType::NCO, 774
OEChem::OEMMFFType::NCP, 780
OEChem::OEMMFFType::NCS, 774
OEChem::OEMMFFType::NGD_, 782
OEChem::OEMMFFType::NIM, 784
OEChem::OEMMFFType::NM, 782
OEChem::OEMMFFType::NN, 774
OEChem::OEMMFFType::NN_, 782
OEChem::OEMMFFType::NNC, 774
OEChem::OEMMFFType::NNN, 774
OEChem::OEMMFFType::NO, 781
OEChem::OEMMFFType::NO2, 781
OEChem::OEMMFFType::NO3, 781
OEChem::OEMMFFType::NPD_, 782
OEChem::OEMMFFType::NPO2, 786
OEChem::OEMMFFType::NPO3, 786
OEChem::OEMMFFType::NPOX, 783
OEChem::OEMMFFType::NPYD, 780
OEChem::OEMMFFType::NPYL, 780
OEChem::OEMMFFType::NR, 774
OEChem::OEMMFFType::NR_, 779
OEChem::OEMMFFType::NR__, 782
OEChem::OEMMFFType::NSO, 781
OEChem::OEMMFFType::NSO2, 781
OEChem::OEMMFFType::NSO3, 781
OEChem::OEMMFFType::NSP, 780
OEChem::OEMMFFType::O, 772
OEChem::OEMMFFType::O2CM, 778
OEChem::OEMMFFType::O2N, 778
OEChem::OEMMFFType::O2NO, 778
OEChem::OEMMFFType::O2P, 779
OEChem::OEMMFFType::O2S, 778
OEChem::OEMMFFType::O3N, 778
OEChem::OEMMFFType::O3P, 779
OEChem::OEMMFFType::O3S, 778
OEChem::OEMMFFType::O4CL, 779
OEChem::OEMMFFType::O4Cl, 779
OEChem::OEMMFFType::O4P, 779
OEChem::OEMMFFType::O4S, 779
OEChem::OEMMFFType::O_, 781
OEChem::OEMMFFType::O__, 781
OEChem::OEMMFFType::OC, 773
OEChem::OEMMFFType::OCC, 772
OEChem::OEMMFFType::OCN, 773
OEChem::OEMMFFType::OCN_, 774
OEChem::OEMMFFType::OCO, 772
OEChem::OEMMFFType::OCO_, 774
OEChem::OEMMFFType::OCR, 774
OEChem::OEMMFFType::OCS, 773
OEChem::OEMMFFType::OFUR, 782
OEChem::OEMMFFType::OH2, 783
OEChem::OEMMFFType::OM, 779
OEChem::OEMMFFType::OM2, 779
OEChem::OEMMFFType::ON, 774

OEChem::OEMMFFType::ON_, 778
OEChem::OEMMFFType::ONO, 773
OEChem::OEMMFFType::ONO2, 773
OEChem::OEMMFFType::OP, 773
OEChem::OEMMFFType::OP_, 779
OEChem::OEMMFFType::OPO, 773
OEChem::OEMMFFType::OPO2, 773
OEChem::OEMMFFType::OPO3, 773
OEChem::OEMMFFType::OR, 772
OEChem::OEMMFFType::OS, 773
OEChem::OEMMFFType::OS_, 774
OEChem::OEMMFFType::OS__, 774
OEChem::OEMMFFType::OS___, 778
OEChem::OEMMFFType::OSMS, 779
OEChem::OEMMFFType::OSO, 773
OEChem::OEMMFFType::OSO2, 773
OEChem::OEMMFFType::OSO3, 773
OEChem::OEMMFFType::OSO_, 773
OEChem::OEMMFFType::OXN, 778
OEChem::OEMMFFType::P, 777
OEChem::OEMMFFType::PC, 784
OEChem::OEMMFFType::PO, 777
OEChem::OEMMFFType::PO2, 777
OEChem::OEMMFFType::PO3, 776
OEChem::OEMMFFType::PO4, 776
OEChem::OEMMFFType::PTET, 777
OEChem::OEMMFFType::S, 775
OEChem::OEMMFFType::S2CM, 786
OEChem::OEMMFFType::SC, 775
OEChem::OEMMFFType::SE, 786
OEChem::OEMMFFType::Se, 786
OEChem::OEMMFFType::SI, 776
OEChem::OEMMFFType::SM, 783
OEChem::OEMMFFType::SN, 775
OEChem::OEMMFFType::SNO, 776
OEChem::OEMMFFType::SO, 775
OEChem::OEMMFFType::SO2, 775
OEChem::OEMMFFType::SO2_, 775
OEChem::OEMMFFType::SO2M, 783
OEChem::OEMMFFType::SO2N, 775
OEChem::OEMMFFType::SO3, 775
OEChem::OEMMFFType::SO4, 775
OEChem::OEMMFFType::SO_, 783
OEChem::OEMMFFType::SP, 783
OEChem::OEMMFFType::SSMO, 783
OEChem::OEMMFFType::SSOM, 783
OEChem::OEMMFFType::STHI, 781
OEChem::OEMMFFType::UNK, 771
OEChem::OEMMFFType::ZN2, 785
OEChem::OEMMFFTypeIndex, 964
OEChem::OEMMFFTypeName, 964
OEChem::OEMMFFTypeNames, 964
OEChem::OEMMFFTypeResult, 550
    Constructors, 550
    GetFailed, 550
    GetMMFFType, 550
OEChem::OEMModType, 787
OEChem::OEMModType::B2, 790
OEChem::OEMModType::B3, 790
OEChem::OEMModType::Ba, 791
OEChem::OEMModType::Br, 791
OEChem::OEMModType::C0, 788
OEChem::OEMModType::C1, 787
OEChem::OEMModType::c1, 792
OEChem::OEMModType::C2, 787
OEChem::OEMModType::c2, 793
OEChem::OEMModType::C3, 787
OEChem::OEMModType::CA, 787
OEChem::OEMModType::Ca, 791
OEChem::OEMModType::CB, 787
OEChem::OEMModType::CC, 787
OEChem::OEMModType::CD, 787
OEChem::OEMModType::CE, 787
OEChem::OEMModType::CF, 787
OEChem::OEMModType::Cl, 791
OEChem::OEMModType::CM, 787
OEChem::OEMModType::Cm, 793
OEChem::OEMModType::CP, 788
OEChem::OEMModType::CR, 788
OEChem::OEMModType::Cs, 791
OEChem::OEMModType::Du, 791
OEChem::OEMModType::F0, 790
OEChem::OEMModType::f2, 792
OEChem::OEMModType::f3, 792
OEChem::OEMModType::H0, 790
OEChem::OEMModType::H1, 790
OEChem::OEMModType::H2, 790
OEChem::OEMModType::H3, 790
OEChem::OEMModType::H4, 790
OEChem::OEMModType::H5, 790
OEChem::OEMModType::I0, 791
OEChem::OEMModType::K0, 791
OEChem::OEMModType::Li, 791
OEChem::OEMModType::Lp, 791
OEChem::OEMModType::M2, 792
OEChem::OEMModType::M3, 792
OEChem::OEMModType::m3, 793
OEChem::OEMModType::M4, 792
OEChem::OEMModType::m4, 793
OEChem::OEMModType::M5, 792
OEChem::OEMModType::m5, 793
OEChem::OEMModType::M6, 792
OEChem::OEMModType::m6, 793
OEChem::OEMModType::M7, 792
OEChem::OEMModType::MAXTYPE, 793
OEChem::OEMModType::Mg, 792
OEChem::OEMModType::N0, 789
OEChem::OEMModType::N1, 788

OEChem::OEMModType::N2, 788
OEChem::OEMModType::n2, 792
OEChem::OEMModType::N3, 788
OEChem::OEMModType::n3, 792
OEChem::OEMModType::N4, 789
OEChem::OEMModType::N5, 789
OEChem::OEMModType::NA, 789
OEChem::OEMModType::Na, 791
OEChem::OEMModType::NB, 789
OEChem::OEMModType::NC, 789
OEChem::OEMModType::ND, 789
OEChem::OEMModType::NE, 789
OEChem::OEMModType::NF, 789
OEChem::OEMModType::NG, 789
OEChem::OEMModType::NH, 789
OEChem::OEMModType::NI, 789
OEChem::OEMModType::NM, 789
OEChem::OEMModType::NP, 789
OEChem::OEMModType::O0, 788
OEChem::OEMModType::O2, 788
OEChem::OEMModType::o2, 792
OEChem::OEMModType::O3, 788
OEChem::OEMModType::o3, 792
OEChem::OEMModType::OA, 788
OEChem::OEMModType::OM, 788
OEChem::OEMModType::OP, 788
OEChem::OEMModType::OQ, 788
OEChem::OEMModType::OW, 788
OEChem::OEMModType::P0, 790
OEChem::OEMModType::Rb, 791
OEChem::OEMModType::S0, 790
OEChem::OEMModType::S1, 790
OEChem::OEMModType::S2, 793
OEChem::OEMModType::SA, 790
OEChem::OEMModType::Si, 791
OEChem::OEMModType::SM, 790
OEChem::OEMModType::SP, 793
OEChem::OEMModType::Z0, 791
OEChem::OEMModType::Zn, 793
OEChem::OEMol, 550
    ClearBase, 552
    ClearMCMol, 552
    ClearMolBase, 552
    Constructors, 551
    MCMol, 552
    SCMol, 552
OEChem::OEMolBase, 552
    Clear, 553
    ClearBase, 553
    ClearCoords, 553
    Compress, 553
    Count, 553
    CreateCopy, 554
    DeleteAtom, 554
    DeleteBond, 554
    DeleteGroup, 554
    GetAtom, 554
    GetAtoms, 555
    GetBond, 555
    GetBonds, 555
    GetCoords, 555
    GetDimension, 556
    GetEnergy, 556
    GetGroup, 556
    GetGroups, 556
    GetMaxAtomIdx, 557
    GetMaxBondIdx, 557
    GetTitle, 557
    HasPerceived, 557
    IsDataType, 558
    IsDeleted, 558
    IsRxn, 558
    NewAtom, 558
    NewBond, 558
    NewGroup, 559
    NumAtoms, 560
    NumBonds, 560
    NumGroups, 560
    operator bool, 553
    OrderAtoms, 560
    OrderBonds, 560
    ResetPerceived, 560
    SetCoords, 560
    SetDimension, 561
    SetEnergy, 561
    SetPerceived, 561
    SetRxn, 562
    SetTitle, 562
    Sweep, 562
    UnCompress, 563
OEChem::OEMolBaseType, 793
OEChem::OEMolBaseType::MaxType, 794
OEChem::OEMolBaseType::OEDBMol, 794
OEChem::OEMolBaseType::OEDefault, 794
OEChem::OEMolBaseType::OEMiniMol, 794
OEChem::OEMolBaseType::Undefined, 793
OEChem::OEMolDatabase, 563
    Clear, 564
    Constructors, 563
    CreateCopy, 564
    Delete, 564
    GetDataType, 564
    GetFlavor, 564
    GetFormat, 564
    GetIdx, 565
    GetIdxs, 565
    GetMaxMolIdx, 565
    GetMolecule, 565

GetMoleculeString, 565
GetRow, 566
GetTitle, 566
GetTitles, 566
IsDataType, 566
IsDeleted, 566
NumMols, 566
Open, 567
Order, 567
Save, 567
WriteMolecule, 568
OEChem::OEMolDatabaseSaveOptions, 568
Constructors, 568
GetFlavor, 569
GetFormat, 569
GetIdxFileName, 569
GetWriteIdx, 569
SetFlavor, 569
SetFormat, 569
SetIdxFileName, 569
SetWriteIdx, 570
OEChem::OEMolDBException, 570
Constructors, 570
what, 570
OEChem::OEMolecularFormula, 964
OEChem::OEMolGDPropertyFxn, 570
Constructors, 570
OEChem::oemolistream, 570
close, 574
Constructors, 571
eof, 574
getbyte, 574
GetConfTest, 571
getline, 574
GetMCMolBases, 572
GetMolBases, 572
GetOEGraphMols, 572
GetReadFailureCt, 572
IncrementReadFailureCt, 572
OpenByteArray, 574
operator bool, 571
peekbyte, 575
read, 575
ResetReadFailureCt, 572
rewind, 575
seek, 575
SetConfTest, 572
SetFlavor, 573
SetFormat, 573
Setgz, 573
SetTracer, 573
size, 575
skip, 575
skipbyte, 575
skipline, 575
tell, 575
OEChem::oemolithread, 576
close, 577
Constructors, 576
GetIStream, 576
GetMolStream, 576
openstring, 577
SetFlavor, 576
SetFormat, 576
OEChem::oemolostream, 577
close, 579
Constructors, 577
flush, 579
GetByteArray, 578
openstring, 579
operator bool, 578
put, 579
SetFlavor, 578
SetFormat, 578
Setgz, 578
SetString, 578
tell, 579
write, 579
OEChem::oemolothread, 579
close, 580
Constructors, 580
GetMolStream, 580
GetOStream, 580
GetString, 580
openstring, 580
SetFlavor, 580
SetFormat, 580
OEChem::OEMolParameter, 580
Constructors, 581
GetSetting, 581
GetValue, 582
SetValue, 582
OEChem::OEMolPropertyFxn, 584
CreateCopy, 584
Get, 584
Has, 585
OEChem::OEMolPropertyList, 585
Add, 585
Clear, 585
Constructors, 585
CreateCopy, 585
GetProperty, 585
GetSize, 585
HasProperties, 586
OEChem::OEMolReaderParameter, 582
Constructors, 583
OEChem::OEMolSDPropertyFxn, 586
Constructors, 586

OEChem::oemolstreambase, 586
ClearCSVHeader, 587
close, 588
GetBinaryIOHandler, 587
GetCSVHeader, 587
GetFileName, 587
GetFlavor, 587
GetFormat, 587
Getgz, 588
open, 589
SetFlavor, 588
SetFormat, 588
OEChem::OEMolStyleColorer, 589
AddColor, 590
ClearColors, 590
Constructors, 589
GetColor, 590
GetColorScheme, 590
GetDataTypes, 590
GetDefaultColor, 591
GetNumColors, 591
GetPropertyName, 591
GetValues, 591
SetDefaultColor, 591
OEChem::OEMolTaggedPropertyFxn, 591
Constructors, 592
OEChem::oemolthreadbase, 592
GetMol, 592
operator bool, 592
PeekMol, 593
PutMol, 593
OEChem::OEMolToCXSMILES, 965
OEChem::OEMolToHelm, 1053
OEChem::OEMolToInChI, 965
OEChem::OEMolToInChIKey, 965
OEChem::OEMolToSmiles, 966
OEChem::OEMolToSTDInChI, 967
OEChem::OEMolToSTDInChIKey, 966
OEChem::OEMolWriterParameter, 583
Constructors, 584
OEChem::OEMonomer, 1026
Constructors, 1026
GetAminoAcidType, 1027
GetCanonicalSmiles, 1027
GetCode, 1027
GetCodes, 1027
GetMonomerType, 1027
GetName, 1028
GetPolymerType, 1028
GetSmiles, 1028
HasCode, 1028
HasRGroup, 1028
IsCTerminal, 1028
IsNTerminal, 1028
IsValid, 1029
NumCodes, 1029
NumRGroups, 1029
OEChem::OEMonomerConnectionSmilesFromCXSmiles, 1052
OEChem::OEMonomerData, 1029
AddCode, 1030
Constructors, 1029
GetCanonicalSmiles, 1031
GetCode, 1030
GetCodes, 1030
GetMonomerType, 1030
GetName, 1030
GetPolymerType, 1031
GetSmiles, 1031
HasCode, 1031
IsValid, 1031
NumCodes, 1032
SetCodes, 1032
SetMonomerType, 1032
SetName, 1032
SetPolymerType, 1032
SetSmiles, 1033
Validate, 1033
OEChem::OEMonomerSet, 1033
AddCodeSet, 1034
AddMonomer, 1034
Constructors, 1034
GetCodes, 1035
GetCodeSets, 1035
GetMonomer, 1035
GetMonomers, 1035
GetPrimaryCodeSet, 1035
GetVersion, 1035
HasCodeInCodeSet, 1036
HasCodeSet, 1036
HasMonomer, 1036
IsValid, 1036
NumCodeSets, 1036
NumMonomers, 1036
SetVersion, 1037
OEChem::OEMonomerType, 665
OEChem::OEMonomerType::Backbone, 665
OEChem::OEMonomerType::Default, 665
OEChem::OEMonomerType::Terminal, 665
OEChem::OEMonomerType::Undefined, 665
OEChem::OEMonomerTypeToString, 1054
OEChem::OEMonomerValidationResult, 1037
Constructors, 1038
GetReturnCode, 1038
GetWarning, 1038
OEChem::OEMonomerValidationReturnCode, 666
OEChem::OEMonomerValidationReturnCode::InvalidMonomerCode, 666

OEChem::OEMonomerValidationReturnCode::InvalidMonomerRGroups, 666
OEChem::OEMonomerValidationReturnCode::InvalidMonomerSMILES, 666
OEChem::OEMonomerValidationReturnCode::InvalidMonomerType, 666
OEChem::OEMonomerValidationReturnCode::InvalidPolymerType, 667
OEChem::OEMonomerValidationReturnCode::MissingMonomerCode, 667
OEChem::OEMonomerValidationReturnCode::MissingMonomerRGroups, 667
OEChem::OEMonomerValidationReturnCode::MissingMonomerSMILES, 667
OEChem::OEMonomerValidationReturnCode::Success, 666
OEChem::OEMonomerValidationReturnCode::UnspecifiedAtomStereo, 667
OEChem::OEMonomerValidationReturnCode::UnspecifiedBondStereo, 667
OEChem::OEMultipleBondCount, 967
OEChem::OEMutableCoords, 593
    Constructors, 594
    GetPtr, 594
OEChem::OENbrs, 594
    Constructors, 594
    GetBgn, 594
    GetDist, 594
    GetDist2, 594
    GetEnd, 595
OEChem::OENearestNbrs, 595
    Constructors, 595
    GetNbrs, 595
OEChem::OENearestNbrsMethod, 794
OEChem::OENearestNbrsMethod::AllByAll, 795
OEChem::OENearestNbrsMethod::Auto, 795
OEChem::OENearestNbrsMethod::Default, 794
OEChem::OENearestNbrsMethod::LocalHash, 795
OEChem::OENearestNbrsMethod::ZCoordSort, 795
OEChem::OENetCharge, 967
OEChem::OENewMCMolBase, 967
OEChem::OENewMolBase, 968
OEChem::OENewQMolBase, 968
OEChem::OENormalize, 968
OEChem::OENthAtom, 595
    Constructors, 596
    CreateCopy, 596
    operator(), 596
OEChem::OEOFlavor, 795
OEChem::OEOFlavor::CAN, 795
OEChem::OEOFlavor::CAN::AllBonds, 796
OEChem::OEOFlavor::CAN::AtomMaps, 796
OEChem::OEOFlavor::CAN::AtomStereo, 796
OEChem::OEOFlavor::CAN::BondStereo, 796
OEChem::OEOFlavor::CAN::Canonical, 796
OEChem::OEOFlavor::CAN::DEFAULT, 796
OEChem::OEOFlavor::CAN::Default, 796
OEChem::OEOFlavor::CAN::EnhStereo, 796
OEChem::OEOFlavor::CAN::ExtBonds, 796
OEChem::OEOFlavor::CAN::Hydrogens, 797
OEChem::OEOFlavor::CAN::ImpHCount, 797
OEChem::OEOFlavor::CAN::Isotopes, 797
OEChem::OEOFlavor::CAN::Kekule, 797
OEChem::OEOFlavor::CAN::RGroups, 797
OEChem::OEOFlavor::CAN::SmiMask, 797
OEChem::OEOFlavor::CAN::SuperAtoms, 797
OEChem::OEOFlavor::CDX, 797
OEChem::OEOFlavor::CDX::Add2D, 797
OEChem::OEOFlavor::CDX::Default, 798
OEChem::OEOFlavor::CDX::UnsetBad2DStereo, 798
OEChem::OEOFlavor::CIF, 798
OEChem::OEOFlavor::CIF::Default, 798
OEChem::OEOFlavor::CSV, 798
OEChem::OEOFlavor::CSV::DEFAULT, 799
OEChem::OEOFlavor::CSV::Default, 798
OEChem::OEOFlavor::CSV::Header, 798
OEChem::OEOFlavor::CXSMILES, 799
OEChem::OEOFlavor::CXSMILES::AllBonds, 799
OEChem::OEOFlavor::CXSMILES::AtomMaps, 799
OEChem::OEOFlavor::CXSMILES::AtomStereo, 799
OEChem::OEOFlavor::CXSMILES::BondStereo, 799
OEChem::OEOFlavor::CXSMILES::Canonical, 799
OEChem::OEOFlavor::CXSMILES::DEFAULT, 799
OEChem::OEOFlavor::CXSMILES::Default, 800
OEChem::OEOFlavor::CXSMILES::EnhStereo, 800
OEChem::OEOFlavor::CXSMILES::ExtBonds, 800
OEChem::OEOFlavor::CXSMILES::Hydrogens, 800
OEChem::OEOFlavor::CXSMILES::ImpHCount, 800
OEChem::OEOFlavor::CXSMILES::Isotopes, 800
OEChem::OEOFlavor::CXSMILES::Kekule, 800
OEChem::OEOFlavor::CXSMILES::RGroups, 800
OEChem::OEOFlavor::CXSMILES::SmiMask, 800
OEChem::OEOFlavor::CXSMILES::SuperAtoms, 800
OEChem::OEOFlavor::FASTA, 801
OEChem::OEOFlavor::FASTA::Default, 801
OEChem::OEOFlavor::Generic, 801
OEChem::OEOFlavor::Generic::AroMask, 801
OEChem::OEOFlavor::Generic::DEFAULT, 802
OEChem::OEOFlavor::Generic::Default, 802
OEChem::OEOFlavor::Generic::GenericMask, 802
OEChem::OEOFlavor::Generic::OEAroModelDaylight, 801
OEChem::OEOFlavor::Generic::OEAroModelMDL, 801
OEChem::OEOFlavor::Generic::OEAroModelMMFF, 801
OEChem::OEOFlavor::Generic::OEAroModelOpenEye, 801

OEChem::OEOFlavor::Generic::OEAroModelTripos, 801
OEChem::OEOFlavor::Generic::Rings, 802
OEChem::OEOFlavor::Generic::SpecificMask, 802
OEChem::OEOFlavor::INCHI, 802
OEChem::OEOFlavor::INCHI::Chiral, 802
OEChem::OEOFlavor::INCHI::Default, 802
OEChem::OEOFlavor::INCHI::FixedHLayer, 803
OEChem::OEOFlavor::INCHI::Hydrogens, 803
OEChem::OEOFlavor::INCHI::RacemicStereo, 803
OEChem::OEOFlavor::INCHI::ReconnectedMetals, 803
OEChem::OEOFlavor::INCHI::RelativeStereo, 803
OEChem::OEOFlavor::INCHI::Stereo, 803
OEChem::OEOFlavor::INCHI::Title, 803
OEChem::OEOFlavor::INCHIKEY, 803
OEChem::OEOFlavor::INCHIKEY::Chiral, 803
OEChem::OEOFlavor::INCHIKEY::Default, 804
OEChem::OEOFlavor::INCHIKEY::FixedHLayer, 804
OEChem::OEOFlavor::INCHIKEY::Hydrogens, 804
OEChem::OEOFlavor::INCHIKEY::RacemicStereo, 804
OEChem::OEOFlavor::INCHIKEY::ReconnectedMetals, 804
OEChem::OEOFlavor::INCHIKEY::RelativeStereo, 804
OEChem::OEOFlavor::INCHIKEY::Stereo, 804
OEChem::OEOFlavor::INCHIKEY::Title, 804
OEChem::OEOFlavor::ISM, 804
OEChem::OEOFlavor::ISM::AllBonds, 805
OEChem::OEOFlavor::ISM::AtomMaps, 805
OEChem::OEOFlavor::ISM::AtomStereo, 805
OEChem::OEOFlavor::ISM::BondStereo, 805
OEChem::OEOFlavor::ISM::Canonical, 805
OEChem::OEOFlavor::ISM::DEFAULT, 805
OEChem::OEOFlavor::ISM::Default, 805
OEChem::OEOFlavor::ISM::EnhStereo, 805
OEChem::OEOFlavor::ISM::ExtBonds, 805
OEChem::OEOFlavor::ISM::Hydrogens, 806
OEChem::OEOFlavor::ISM::ImpHCount, 806
OEChem::OEOFlavor::ISM::Isotopes, 806
OEChem::OEOFlavor::ISM::Kekule, 806
OEChem::OEOFlavor::ISM::RGroups, 806
OEChem::OEOFlavor::ISM::SmiMask, 806
OEChem::OEOFlavor::ISM::SuperAtoms, 806
OEChem::OEOFlavor::MDL, 806
OEChem::OEOFlavor::MDL::Add2D, 806
OEChem::OEOFlavor::MDL::CurrentParity, 808
OEChem::OEOFlavor::MDL::DEFAULT, 808
OEChem::OEOFlavor::MDL::Default, 808
OEChem::OEOFlavor::MDL::MCHG, 807
OEChem::OEOFlavor::MDL::MDLParity, 807
OEChem::OEOFlavor::MDL::MISO, 807
OEChem::OEOFlavor::MDL::MMask, 808
OEChem::OEOFlavor::MDL::MRGP, 807
OEChem::OEOFlavor::MDL::MV30, 807
OEChem::OEOFlavor::MDL::NoParity, 807
OEChem::OEOFlavor::MDL::PMask, 808
OEChem::OEOFlavor::MDL::SuppressImp2ExpENHSTE, 808
OEChem::OEOFlavor::MDL::SuppressTimestamps, 808
OEChem::OEOFlavor::MDL::UnsetBad2DStereo, 806
OEChem::OEOFlavor::MF, 809
OEChem::OEOFlavor::MF::DEFAULT, 809
OEChem::OEOFlavor::MF::Default, 809
OEChem::OEOFlavor::MF::Title, 809
OEChem::OEOFlavor::MMCIF, 809
OEChem::OEOFlavor::MMCIF::Default, 809
OEChem::OEOFlavor::MMCIF::PerceiveMetadata, 809
OEChem::OEOFlavor::MMOD, 809
OEChem::OEOFlavor::MMOD::AtomTypes, 809
OEChem::OEOFlavor::MMOD::DEFAULT, 809
OEChem::OEOFlavor::MMOD::Default, 810
OEChem::OEOFlavor::MOL2, 810
OEChem::OEOFlavor::MOL2::AllMask, 811
OEChem::OEOFlavor::MOL2::AtomNames, 810
OEChem::OEOFlavor::MOL2::AtomTypeNames, 810
OEChem::OEOFlavor::MOL2::BondTypeNames, 810
OEChem::OEOFlavor::MOL2::ChargePrecision, 810
OEChem::OEOFlavor::MOL2::DEFAULT, 811
OEChem::OEOFlavor::MOL2::Default, 811
OEChem::OEOFlavor::MOL2::Forcefield, 810
OEChem::OEOFlavor::MOL2::GeneralFFFormat, 810
OEChem::OEOFlavor::MOL2::Hydrogens, 810
OEChem::OEOFlavor::MOL2::NameMask, 811
OEChem::OEOFlavor::MOL2::OrderAtoms, 811
OEChem::OEOFlavor::MOL2::Substructure, 811
OEChem::OEOFlavor::MOL2H, 811
OEChem::OEOFlavor::MOL2H::AllMask, 812
OEChem::OEOFlavor::MOL2H::AtomNames, 811
OEChem::OEOFlavor::MOL2H::AtomTypeNames, 811
OEChem::OEOFlavor::MOL2H::BondTypeNames, 811
OEChem::OEOFlavor::MOL2H::DEFAULT, 812
OEChem::OEOFlavor::MOL2H::Default, 812
OEChem::OEOFlavor::MOL2H::Hydrogens, 811
OEChem::OEOFlavor::MOL2H::NameMask, 812
OEChem::OEOFlavor::MOL2H::OrderAtoms, 811
OEChem::OEOFlavor::MOL2H::Substructure, 812
OEChem::OEOFlavor::MOPAC, 812
OEChem::OEOFlavor::MOPAC::AllMask, 812
OEChem::OEOFlavor::MOPAC::CHARGES, 812
OEChem::OEOFlavor::MOPAC::DEFAULT, 812
OEChem::OEOFlavor::MOPAC::Default, 812
OEChem::OEOFlavor::MOPAC::XYZ, 812
OEChem::OEOFlavor::OEB, 812
OEChem::OEOFlavor::OEB::Default, 812

OEChem::OEOFlavor::OEZ, 812
OEChem::OEOFlavor::OEZ::Default, 812
OEChem::OEOFlavor::PDB, 813
OEChem::OEOFlavor::PDB::AllMask, 815
OEChem::OEOFlavor::PDB::BasicMask, 814
OEChem::OEOFlavor::PDB::BONDS, 813
OEChem::OEOFlavor::PDB::BOTH, 813
OEChem::OEOFlavor::PDB::CHARGE, 813
OEChem::OEOFlavor::PDB::CurrentResidues, 815
OEChem::OEOFlavor::PDB::DEFAULT, 815
OEChem::OEOFlavor::PDB::Default, 815
OEChem::OEOFlavor::PDB::DELPHI, 813
OEChem::OEOFlavor::PDB::ELEMENT, 814
OEChem::OEOFlavor::PDB::FORMALCHARGE, 814
OEChem::OEOFlavor::PDB::HETBONDS, 814
OEChem::OEOFlavor::PDB::NoResidues, 814
OEChem::OEOFlavor::PDB::OEResidues, 814
OEChem::OEOFlavor::PDB::OrderAtoms, 815
OEChem::OEOFlavor::PDB::ORDERS, 813
OEChem::OEOFlavor::PDB::RADIUS, 813
OEChem::OEOFlavor::PDB::TER, 813
OEChem::OEOFlavor::SDF, 815
OEChem::OEOFlavor::SDF::Add2D, 815
OEChem::OEOFlavor::SDF::CurrentParity, 816
OEChem::OEOFlavor::SDF::DEFAULT, 817
OEChem::OEOFlavor::SDF::Default, 817
OEChem::OEOFlavor::SDF::MCHG, 816
OEChem::OEOFlavor::SDF::MDLParity, 816
OEChem::OEOFlavor::SDF::MISO, 816
OEChem::OEOFlavor::SDF::MMask, 817
OEChem::OEOFlavor::SDF::MRGP, 816
OEChem::OEOFlavor::SDF::MV30, 816
OEChem::OEOFlavor::SDF::NoParity, 816
OEChem::OEOFlavor::SDF::PMask, 817
OEChem::OEOFlavor::SDF::SuppressImp2ExpENHSTE, 817
OEChem::OEOFlavor::SDF::SuppressTimestamps, 817
OEChem::OEOFlavor::SDF::UnsetBad2DStereo, 815
OEChem::OEOFlavor::SLN, 817
OEChem::OEOFlavor::SLN::Default, 817
OEChem::OEOFlavor::SMI, 818
OEChem::OEOFlavor::SMI::AllBonds, 818
OEChem::OEOFlavor::SMI::AtomMaps, 818
OEChem::OEOFlavor::SMI::AtomStereo, 818
OEChem::OEOFlavor::SMI::BondStereo, 818
OEChem::OEOFlavor::SMI::Canonical, 818
OEChem::OEOFlavor::SMI::DEFAULT, 818
OEChem::OEOFlavor::SMI::Default, 818
OEChem::OEOFlavor::SMI::EnhStereo, 818
OEChem::OEOFlavor::SMI::ExtBonds, 819
OEChem::OEOFlavor::SMI::Hydrogens, 819
OEChem::OEOFlavor::SMI::ImpHCount, 819
OEChem::OEOFlavor::SMI::Isotopes, 819
OEChem::OEOFlavor::SMI::Kekule, 819
OEChem::OEOFlavor::SMI::RGroups, 819
OEChem::OEOFlavor::SMI::SmiMask, 819
OEChem::OEOFlavor::SMI::SuperAtoms, 819
OEChem::OEOFlavor::USM, 819
OEChem::OEOFlavor::USM::AllBonds, 819
OEChem::OEOFlavor::USM::AtomMaps, 819
OEChem::OEOFlavor::USM::AtomStereo, 820
OEChem::OEOFlavor::USM::BondStereo, 820
OEChem::OEOFlavor::USM::Canonical, 820
OEChem::OEOFlavor::USM::DEFAULT, 820
OEChem::OEOFlavor::USM::Default, 820
OEChem::OEOFlavor::USM::EnhStereo, 820
OEChem::OEOFlavor::USM::ExtBonds, 820
OEChem::OEOFlavor::USM::Hydrogens, 820
OEChem::OEOFlavor::USM::ImpHCount, 820
OEChem::OEOFlavor::USM::Isotopes, 820
OEChem::OEOFlavor::USM::Kekule, 821
OEChem::OEOFlavor::USM::RGroups, 821
OEChem::OEOFlavor::USM::SmiMask, 821
OEChem::OEOFlavor::USM::SuperAtoms, 821
OEChem::OEOFlavor::XYZ, 821
OEChem::OEOFlavor::XYZ::Charges, 821
OEChem::OEOFlavor::XYZ::Default, 821
OEChem::OEOFlavor::XYZ::DoublePrecision, 821
OEChem::OEOFlavor::XYZ::SinglePrecision, 821
OEChem::OEOFlavor::XYZ::Symbols, 821
OEChem::OEOmegaConfTest, 596
    CombineMols, 597
    CompareMols, 597
    Constructors, 597
    CreateCopy, 597
OEChem::OEOutFileStringParameter, 597
    Constructors, 598
OEChem::OEOverlayConfsByRMSD, 968
OEChem::OEParseInChI, 969
OEChem::OEParseSmarts, 969
OEChem::OEParseSmiles, 970
OEChem::OEParseSmilesOptions, 598
    Constructors, 599
    GetCanon, 599
    GetCXSMILES, 599
    GetQuiet, 599
    GetStrict, 599
    SetCanon, 599
    SetCXSMILES, 600
    SetQuiet, 600
    SetStrict, 600
OEChem::OEParseSmirks, 971
OEChem::OEPartPredT, 600
    Constructors, 601
    CreateCopy, 601
    operator(), 601
    SelectPart, 601

OEChem::OEPDBAtomName, 822
OEChem::OEPDBAtomName::C, 822
OEChem::OEPDBAtomName::C10, 830
OEChem::OEPDBAtomName::C11, 830
OEChem::OEPDBAtomName::C12, 830
OEChem::OEPDBAtomName::C13, 830
OEChem::OEPDBAtomName::C14, 830
OEChem::OEPDBAtomName::C15, 830
OEChem::OEPDBAtomName::C16, 830
OEChem::OEPDBAtomName::C19, 831
OEChem::OEPDBAtomName::C1_, 828
OEChem::OEPDBAtomName::C2, 828
OEChem::OEPDBAtomName::C21, 831
OEChem::OEPDBAtomName::C24, 831
OEChem::OEPDBAtomName::C2_, 827
OEChem::OEPDBAtomName::C2A, 827
OEChem::OEPDBAtomName::C3, 829
OEChem::OEPDBAtomName::C3_, 827
OEChem::OEPDBAtomName::C4, 829
OEChem::OEPDBAtomName::C4_, 827
OEChem::OEPDBAtomName::C5, 828
OEChem::OEPDBAtomName::C5_, 827
OEChem::OEPDBAtomName::C5M, 829
OEChem::OEPDBAtomName::C6, 828
OEChem::OEPDBAtomName::C8, 828
OEChem::OEPDBAtomName::CA, 822
OEChem::OEPDBAtomName::CAnext, 832
OEChem::OEPDBAtomName::CB, 822
OEChem::OEPDBAtomName::CD, 823
OEChem::OEPDBAtomName::CD1, 823
OEChem::OEPDBAtomName::CD2, 824
OEChem::OEPDBAtomName::CE, 824
OEChem::OEPDBAtomName::CE1, 824
OEChem::OEPDBAtomName::CE2, 825
OEChem::OEPDBAtomName::CE3, 825
OEChem::OEPDBAtomName::CG, 822
OEChem::OEPDBAtomName::CG1, 823
OEChem::OEPDBAtomName::CG2, 823
OEChem::OEPDBAtomName::CH1, 826
OEChem::OEPDBAtomName::CH2, 826
OEChem::OEPDBAtomName::CH3, 826
OEChem::OEPDBAtomName::CM1, 830
OEChem::OEPDBAtomName::CM2, 830
OEChem::OEPDBAtomName::CM5, 829
OEChem::OEPDBAtomName::CM7, 829
OEChem::OEPDBAtomName::Cprev, 831
OEChem::OEPDBAtomName::CZ, 825
OEChem::OEPDBAtomName::CZ2, 825
OEChem::OEPDBAtomName::CZ3, 826
OEChem::OEPDBAtomName::N, 822
OEChem::OEPDBAtomName::N1, 828
OEChem::OEPDBAtomName::N2, 829
OEChem::OEPDBAtomName::N20, 831
OEChem::OEPDBAtomName::N3, 829
OEChem::OEPDBAtomName::N4, 829
OEChem::OEPDBAtomName::N6, 828
OEChem::OEPDBAtomName::N7, 828
OEChem::OEPDBAtomName::N9, 828
OEChem::OEPDBAtomName::ND1, 823
OEChem::OEPDBAtomName::ND2, 824
OEChem::OEPDBAtomName::NE, 824
OEChem::OEPDBAtomName::NE1, 824
OEChem::OEPDBAtomName::NE2, 825
OEChem::OEPDBAtomName::NH1, 826
OEChem::OEPDBAtomName::NH2, 826
OEChem::OEPDBAtomName::Nnext, 831
OEChem::OEPDBAtomName::NZ, 825
OEChem::OEPDBAtomName::O, 822
OEChem::OEPDBAtomName::O17, 831
OEChem::OEPDBAtomName::O18, 831
OEChem::OEPDBAtomName::O1P, 826
OEChem::OEPDBAtomName::O2, 829
OEChem::OEPDBAtomName::O22, 831
OEChem::OEPDBAtomName::O23, 831
OEChem::OEPDBAtomName::O2_, 827
OEChem::OEPDBAtomName::O2P, 827
OEChem::OEPDBAtomName::O3_, 827
OEChem::OEPDBAtomName::O3P, 830
OEChem::OEPDBAtomName::O4, 829
OEChem::OEPDBAtomName::O4_, 827
OEChem::OEPDBAtomName::O5_, 827
OEChem::OEPDBAtomName::O6, 828
OEChem::OEPDBAtomName::OD, 823
OEChem::OEPDBAtomName::OD1, 824
OEChem::OEPDBAtomName::OD2, 824
OEChem::OEPDBAtomName::OE, 824
OEChem::OEPDBAtomName::OE1, 825
OEChem::OEPDBAtomName::OE2, 825
OEChem::OEPDBAtomName::OE3, 825
OEChem::OEPDBAtomName::OE4, 825
OEChem::OEPDBAtomName::OG, 823
OEChem::OEPDBAtomName::OG1, 823
OEChem::OEPDBAtomName::OH, 826
OEChem::OEPDBAtomName::OXT, 826
OEChem::OEPDBAtomName::P, 826
OEChem::OEPDBAtomName::SD, 823
OEChem::OEPDBAtomName::SE_, 824
OEChem::OEPDBAtomName::SG, 823
OEChem::OEPDBAtomName::UNK, 822
OEChem::OEPDBOrderAtoms, 972
OEChem::OEPerceiveBondOrders, 972
OEChem::OEPerceiveChiral, 972
OEChem::OEPerceiveCIPStereo, 973
OEChem::OEPerceived, 832
OEChem::OEPerceived::Aromaticity, 832
OEChem::OEPerceived::AtomStereo, 832
OEChem::OEPerceived::BondStereo, 832
OEChem::OEPerceived::Chains, 832

OEChem::OEPerceived::Chiral, 832
OEChem::OEPerceived::Energy, 832
OEChem::OEPerceived::Hybridization, 833
OEChem::OEPerceived::PartialCharges, 833
OEChem::OEPerceived::RingAtomsAndBonds, 833
OEChem::OEPerceived::SecondaryStructure, 833
OEChem::OEPerceivePeptideBackbone, 1054
OEChem::OEPerceivePeptideMonomerConnections, 1055
OEChem::OEPerceiveResidues, 973
OEChem::OEPerceiveSymmetry, 974
OEChem::OEPolymerType, 667
OEChem::OEPolymerType::Chem, 667
OEChem::OEPolymerType::Default, 667
OEChem::OEPolymerType::Peptide, 668
OEChem::OEPolymerType::Undefined, 668
OEChem::OEPolymerTypeToString, 1055
OEChem::OEPRECompress, 974
OEChem::OEPredicateParameter, 601
  Constructors, 602
  GetDefault, 602
  GetSetting, 603
  GetValue, 603
  SetDefault, 603
  SetValue, 603
OEChem::OEPrepareSearch, 974
OEChem::OEPreserveResInfo, 833
OEChem::OEPreserveResInfo::All, 833
OEChem::OEPreserveResInfo::AlternateLocation, 833
OEChem::OEPreserveResInfo::AtomName, 833
OEChem::OEPreserveResInfo::BFactor, 833
OEChem::OEPreserveResInfo::ChainID, 833
OEChem::OEPreserveResInfo::Default, 833
OEChem::OEPreserveResInfo::FragmentNumber, 833
OEChem::OEPreserveResInfo::HetAtom, 834
OEChem::OEPreserveResInfo::InsertCode, 834
OEChem::OEPreserveResInfo::None, 834
OEChem::OEPreserveResInfo::Occupancy, 834
OEChem::OEPreserveResInfo::ResidueName, 834
OEChem::OEPreserveResInfo::ResidueNumber, 834
OEChem::OEPreserveResInfo::SerialNumber, 834
OEChem::OEPreserveRotCompress, 975
OEChem::OEProperty, 834
OEChem::OEProperty::Acceptor, 834
OEChem::OEProperty::All, 839
OEChem::OEProperty::Anisou, 840
OEChem::OEProperty::Aromatic, 834
OEChem::OEProperty::AroModel, 839
OEChem::OEProperty::AtomMemberPred, 838
OEChem::OEProperty::AtomName, 838
OEChem::OEProperty::AtomTypeInt, 838
OEChem::OEProperty::AtomTypes, 837
OEChem::OEProperty::AtomTypeString, 837
OEChem::OEProperty::BondMemberPred, 839
OEChem::OEProperty::BondOrders, 837
OEChem::OEProperty::BondStereo, 839
OEChem::OEProperty::BondTypeString, 837
OEChem::OEProperty::Both, 835
OEChem::OEProperty::Chiral, 835
OEChem::OEProperty::Chirality, 836
OEChem::OEProperty::Closure, 836
OEChem::OEProperty::ClosureBonds, 836
OEChem::OEProperty::Component, 839
OEChem::OEProperty::Degree, 836
OEChem::OEProperty::Deleted, 835
OEChem::OEProperty::DeletedAtoms, 837
OEChem::OEProperty::DeletedBonds, 837
OEChem::OEProperty::Dimension, 839
OEChem::OEProperty::Donor, 834
OEChem::OEProperty::Double, 835
OEChem::OEProperty::Endo, 835
OEChem::OEProperty::Exo, 835
OEChem::OEProperty::Hybridization, 836
OEChem::OEProperty::HydrogenMode, 837
OEChem::OEProperty::Isotope, 837
OEChem::OEProperty::Kekule, 836
OEChem::OEProperty::MapIdx, 839
OEChem::OEProperty::MatchOrder, 838
OEChem::OEProperty::Monomer, 840
OEChem::OEProperty::OEBase, 838
OEChem::OEProperty::PartialCharge, 837
OEChem::OEProperty::PartialChargeModel, 839
OEChem::OEProperty::Perceived, 839
OEChem::OEProperty::PhCorrected, 836
OEChem::OEProperty::Radius, 840
OEChem::OEProperty::Residue, 837
OEChem::OEProperty::Ring, 834
OEChem::OEProperty::RingAtomsAndBonds, 836
OEChem::OEProperty::Rotor, 836
OEChem::OEProperty::Rxn, 838
OEChem::OEProperty::RxnRole, 838
OEChem::OEProperty::Single, 835
OEChem::OEProperty::StereoDefined, 835
OEChem::OEProperty::SymClass, 839
OEChem::OEProperty::Triple, 835
OEChem::OEProperty::Undefined, 834
OEChem::OEProperty::Valence, 836
OEChem::OEProperty::Visit, 835
OEChem::OEProperty::X, 838
OEChem::OEProperty::Y, 838
OEChem::OEProperty::Z, 838
OEChem::OEProteinColorScheme, 840
OEChem::OEProteinColorScheme::AtomColor, 840
OEChem::OEProteinColorScheme::BEGIN, 840
OEChem::OEProteinColorScheme::END, 841
OEChem::OEProteinStyle, 841

OEChem::OEProteinStyle::Backbone, 841
OEChem::OEProteinStyle::BEGIN, 841
OEChem::OEProteinStyle::CAlpha, 841
OEChem::OEProteinStyle::END, 842
OEChem::OEProteinStyle::Off, 842
OEChem::OEProteinStyle::Ribbons, 843
OEChem::OEQAtomBase, 603
    GetParentAtom, 604
    GetQAtoms, 604
    GetQBonds, 605
    IsDataType, 605
OEChem::OEQBase, 605
    GetExpr, 605
    SetExpr, 605
OEChem::OEQBondBase, 606
    GetParentBond, 606
    IsDataType, 607
OEChem::OEQMol, 607
    BuildExpressions, 608
    Constructors, 607
    GetQAtoms, 609
    GetQBonds, 609
    QMol, 609
    SCMol, 609
OEChem::OEQMolBase, 609
    BuildExpressions, 610
    Constructors, 610
    CreateCopy, 611
    GetQAtoms, 611
    GetQBonds, 611
    IsDataType, 611
OEChem::OEQMolType, 844
OEChem::OEQMolType::MaxType, 844
OEChem::OEQMolType::OEDefault, 844
OEChem::OEQMolType::Undefined, 844
OEChem::OEQuaternion, 611
    Constructors, 612
    CreateCopy, 612
    Extract, 612
    GetInverse, 612
    GetQuaternion, 612
OEChem::OERadiiType, 844
OEChem::OERadiiType::BondiHVdw, 845
OEChem::OERadiiType::BondiVdw, 845
OEChem::OERadiiType::Covalent, 845
OEChem::OERadiiType::Default, 845
OEChem::OERadiiType::Delphi, 845
OEChem::OERadiiType::Delphi_Robust, 845
OEChem::OERadiiType::HonigIonicCavity, 845
OEChem::OERadiiType::HonigIonicCavity_Robust, 845
OEChem::OERadiiType::Max, 845
OEChem::OERadiiType::None, 846
OEChem::OERadiiType::PaulingVdw, 846
OEChem::OERadiiType::PaulingVdw_Robust, 846
OEChem::OERadiiType::Zap7, 846
OEChem::OERadiiType::Zap7_CarbonylOxygen, 846
OEChem::OERadiiType::Zap7_DoublyBondedOxygen, 846
OEChem::OERadiiType::Zap9, 846
OEChem::OERadiiType::Zap9_JCAMD2010, 846
OEChem::OERadiiType::Zap9_JMC2008, 846
OEChem::OERandomizeTorsions, 976
OEChem::OEReadCDXFile, 976
OEChem::OEReadCIFFile, 976
OEChem::OEReadCSVFile, 977
OEChem::OEReadCSVHeader, 977
OEChem::OEReadFASTAFile, 977
OEChem::OEReadMacroModelFile, 977
OEChem::OEReadMDLFile, 977
OEChem::OEReadMDLQueryFile, 978
OEChem::OEReadMDLReactionQueryFile, 978
OEChem::OEReadMol2File, 978
OEChem::OEReadMolecule, 979
OEChem::OEReadMolFromBytes, 979
OEChem::OEReadMonomerSet, 1055
OEChem::OEReadMOPACFile, 980
OEChem::OEReadOEBFile, 980
OEChem::OEReadPDBFile, 980
OEChem::OEReadRxnFile, 980
OEChem::OEReadSketchFile, 980
OEChem::OEReadXYZFile, 980
OEChem::OERefInputAppOptions, 612
    Constructors, 613
    GetInFile, 613
    GetOutFile, 614
    GetRefFile, 614
    SetInFile, 614
    SetOutFile, 614
    SetRefFile, 614
OEChem::OEResidue, 614
    Constructors, 615
    GetAlternateLocation, 615
    GetBFactor, 615
    GetChainID, 616
    GetEntityID, 617
    GetExtChainID, 616
    GetFragmentNumber, 616
    GetInsertCode, 616
    GetModelNumber, 616
    GetName, 617
    GetOccupancy, 617
    GetResidueNumber, 617
    GetSecondaryStructure, 617
    GetSequenceID, 617
    GetSerialNumber, 617
    GetSubChainID, 616
    IsHetAtom, 618

operator bool, 615
SetAlternateLocation, 618
SetBFactor, 618
SetChainID, 618
SetEntityID, 620
SetExtChainID, 618
SetFragmentNumber, 619
SetHetAtom, 619
SetInsertCode, 619
SetModelNumber, 619
SetName, 619
SetOccupancy, 620
SetResidueNumber, 620
SetSecondaryStructure, 620
SetSequenceID, 620
SetSerialNumber, 620
SetSubChainID, 619
OEChem::OEResidueHydrogens, 981
OEChem::OEResidueIndex, 846
OEChem::OEResidueIndex::_1MA, 850
OEChem::OEResidueIndex::_2MG, 850
OEChem::OEResidueIndex::_3NI, 855
OEChem::OEResidueIndex::_4MO, 856
OEChem::OEResidueIndex::_4TI, 857
OEChem::OEResidueIndex::_5MC, 850
OEChem::OEResidueIndex::_5MU, 850
OEChem::OEResidueIndex::_6MO, 856
OEChem::OEResidueIndex::_7MG, 850
OEChem::OEResidueIndex::A, 849
OEChem::OEResidueIndex::ABA, 851
OEChem::OEResidueIndex::ACE, 851
OEChem::OEResidueIndex::ADP, 853
OEChem::OEResidueIndex::AF3, 853
OEChem::OEResidueIndex::ALA, 847
OEChem::OEResidueIndex::ALF, 853
OEChem::OEResidueIndex::ARG, 847
OEChem::OEResidueIndex::ASH, 848
OEChem::OEResidueIndex::ASN, 847
OEChem::OEResidueIndex::ASP, 847
OEChem::OEResidueIndex::ASX, 848
OEChem::OEResidueIndex::ATP, 853
OEChem::OEResidueIndex::AU, 856
OEChem::OEResidueIndex::AU3, 856
OEChem::OEResidueIndex::BME, 853
OEChem::OEResidueIndex::BR, 852
OEChem::OEResidueIndex::C, 849
OEChem::OEResidueIndex::CGU, 851
OEChem::OEResidueIndex::CIM, 854
OEChem::OEResidueIndex::CIP, 854
OEChem::OEResidueIndex::CL, 852
OEChem::OEResidueIndex::CME, 851
OEChem::OEResidueIndex::CO, 855
OEChem::OEResidueIndex::CO3, 855
OEChem::OEResidueIndex::CSD, 851
OEChem::OEResidueIndex::CU, 855
OEChem::OEResidueIndex::CU1, 855
OEChem::OEResidueIndex::CYH, 849
OEChem::OEResidueIndex::CYM, 849
OEChem::OEResidueIndex::CYS, 847
OEChem::OEResidueIndex::CYX, 848
OEChem::OEResidueIndex::DA, 850
OEChem::OEResidueIndex::DC, 850
OEChem::OEResidueIndex::DG, 850
OEChem::OEResidueIndex::DI, 850
OEChem::OEResidueIndex::DMS, 853
OEChem::OEResidueIndex::DU, 850
OEChem::OEResidueIndex::EDO, 853
OEChem::OEResidueIndex::EOH, 853
OEChem::OEResidueIndex::EU, 856
OEChem::OEResidueIndex::EU3, 856
OEChem::OEResidueIndex::FAD, 853
OEChem::OEResidueIndex::FE, 855
OEChem::OEResidueIndex::FE2, 855
OEChem::OEResidueIndex::FOR, 851
OEChem::OEResidueIndex::G, 849
OEChem::OEResidueIndex::GLH, 849
OEChem::OEResidueIndex::GLN, 847
OEChem::OEResidueIndex::GLU, 847
OEChem::OEResidueIndex::GLX, 848
OEChem::OEResidueIndex::GLY, 847
OEChem::OEResidueIndex::GOL, 853
OEChem::OEResidueIndex::H2U, 850
OEChem::OEResidueIndex::HEM, 853
OEChem::OEResidueIndex::HID, 849
OEChem::OEResidueIndex::HIE, 849
OEChem::OEResidueIndex::HIP, 849
OEChem::OEResidueIndex::HIS, 847
OEChem::OEResidueIndex::HOH, 853
OEChem::OEResidueIndex::HSE, 851
OEChem::OEResidueIndex::HYL, 851
OEChem::OEResidueIndex::HYP, 851
OEChem::OEResidueIndex::I, 849
OEChem::OEResidueIndex::ILE, 847
OEChem::OEResidueIndex::IOD, 853
OEChem::OEResidueIndex::IR, 856
OEChem::OEResidueIndex::IR3, 856
OEChem::OEResidueIndex::LEU, 847
OEChem::OEResidueIndex::LIG, 852
OEChem::OEResidueIndex::LYN, 849
OEChem::OEResidueIndex::LYS, 848
OEChem::OEResidueIndex::LYZ, 851
OEChem::OEResidueIndex::M2G, 850
OEChem::OEResidueIndex::MEN, 851
OEChem::OEResidueIndex::MET, 848
OEChem::OEResidueIndex::MLE, 851
OEChem::OEResidueIndex::MLY, 852
OEChem::OEResidueIndex::MN, 855
OEChem::OEResidueIndex::MN3, 855

OEChem::OEResidueIndex::MOH, 853
OEChem::OEResidueIndex::MOL, 852
OEChem::OEResidueIndex::MSE, 852
OEChem::OEResidueIndex::NAD, 854
OEChem::OEResidueIndex::NAG, 854
OEChem::OEResidueIndex::NHE, 854
OEChem::OEResidueIndex::NI, 855
OEChem::OEResidueIndex::NME, 854
OEChem::OEResidueIndex::OMC, 850
OEChem::OEResidueIndex::OMG, 850
OEChem::OEResidueIndex::ORN, 852
OEChem::OEResidueIndex::OS, 856
OEChem::OEResidueIndex::OS4, 856
OEChem::OEResidueIndex::PCA, 852
OEChem::OEResidueIndex::PEO, 854
OEChem::OEResidueIndex::PER, 854
OEChem::OEResidueIndex::PHE, 848
OEChem::OEResidueIndex::PO2, 854
OEChem::OEResidueIndex::PO4, 854
OEChem::OEResidueIndex::PRO, 848
OEChem::OEResidueIndex::PSU, 851
OEChem::OEResidueIndex::PT, 856
OEChem::OEResidueIndex::PT4, 856
OEChem::OEResidueIndex::PTR, 852
OEChem::OEResidueIndex::RU, 854
OEChem::OEResidueIndex::SAR, 852
OEChem::OEResidueIndex::SB, 855
OEChem::OEResidueIndex::SEP, 852
OEChem::OEResidueIndex::SER, 848
OEChem::OEResidueIndex::SO2, 854
OEChem::OEResidueIndex::SO4, 854
OEChem::OEResidueIndex::SX, 854
OEChem::OEResidueIndex::T, 849
OEChem::OEResidueIndex::TAU, 852
OEChem::OEResidueIndex::THR, 848
OEChem::OEResidueIndex::TPO, 852
OEChem::OEResidueIndex::TRP, 848
OEChem::OEResidueIndex::TYM, 849
OEChem::OEResidueIndex::TYR, 848
OEChem::OEResidueIndex::U, 849
OEChem::OEResidueIndex::UNK, 848
OEChem::OEResidueIndex::UNL, 852
OEChem::OEResidueIndex::V, 855
OEChem::OEResidueIndex::VAL, 848
OEChem::OEResidueIndex::Y1, 855
OEChem::OEResidueIndex::YB, 856
OEChem::OEResidueIndex::YB2, 856
OEChem::OEResidueIndex::YG, 851
OEChem::OEResidueIndex::YT3, 855
OEChem::OEResidueToString, 981
OEChem::OERGroupsToHydrogens, 981
OEChem::OERingBondCount, 981
OEChem::OERMSD, 982
OEChem::OERotate, 986
OEChem::OERotMatrix, 620
    Constructors, 621
    CreateCopy, 621
    Extract, 621
    GetInverse, 621
    GetRotMatrix, 621
OEChem::OERxnAutomapMethod, 663
OEChem::OERxnAutomapMethod::DEFAULT, 664
OEChem::OERxnAutomapMethod::MCSMapping, 664
OEChem::OERxnAutomapper, 1038
    AutoMapRxn, 1039
    Constructors, 1038
    GetOptions, 1039
OEChem::OERxnAutomapperOptions, 1039
    Constructors, 1039
    GetClearMapping, 1040
    GetExtendMappedRegions, 1040
    GetMappingMethod, 1040
    GetMCSExhaustiveMode, 1040
    GetStrictMapping, 1040
    SetClearMapping, 1041
    SetExtendMappedRegions, 1041
    SetMappingMethod, 1041
    SetMCSExhaustiveMode, 1041
    SetStrictMapping, 1041
OEChem::OERxnAutomapStatus, 664
OEChem::OERxnAutomapStatus::Failure, 664
OEChem::OERxnAutomapStatus::InvalidComponents, 664
OEChem::OERxnAutomapStatus::InvalidMethod, 664
OEChem::OERxnAutomapStatus::NonReactionInput, 665
OEChem::OERxnAutomapStatus::Success, 665
OEChem::OERxnRole, 857
OEChem::OERxnRole::Agent, 857
OEChem::OERxnRole::Catalyst, 857
OEChem::OERxnRole::None, 857
OEChem::OERxnRole::Product, 857
OEChem::OERxnRole::Reactant, 857
OEChem::OESameAlternateLocation, 987
OEChem::OESameChain, 987
OEChem::OESameResidue, 987
OEChem::OESameSubSearchScreenTypes, 987
OEChem::OEScrambleMolecule, 988
OEChem::OEScreen, 988
OEChem::OESDDataPair, 621
    Constructors, 621
    GetTag, 622
    GetValue, 622
    SetTag, 622
    SetValue, 622
OEChem::OESet2DHydrogenGeom, 989
OEChem::OESet3DHydrogenGeom, 989

OEChem::OESetAngle, 989
OEChem::OESetAnisou, 990
OEChem::OESetAtomComment, 990
OEChem::OESetCIPStereo, 990
OEChem::OESetComment, 991
OEChem::OESetDefaultFlavors, 991
OEChem::OESetDefaultIFlavors, 991
OEChem::OESetDefaultOFlavors, 992
OEChem::OESetDimensionFromCoords, 992
OEChem::OESetDistance, 993
OEChem::OESetMCSExhaustiveSearchTruncationLimit, 993
OEChem::OESetMDLDimensionFromCoords, 993
OEChem::OESetMMCIFChemCompData, 994
OEChem::OESetMMCIFData, 995
OEChem::OESetOEGraphMol, 995
OEChem::OESetOEMol, 996
OEChem::OESetOEQMol, 996
OEChem::OESetPackedCoords, 996
OEChem::OESetPDBData, 996
OEChem::OESetSDData, 997
OEChem::OESetStyle, 997
OEChem::OESetStyleIndex, 997
OEChem::OESetTorsion, 998
OEChem::OEShortestPath, 998
OEChem::OESimilarByRMSD, 622
  Constructors, 622
  CreateCopy, 623
  operator(), 622
OEChem::OESimpleAppOptions, 623
  Constructors, 623
  GetInFile, 624
  GetOutFile, 624
  SetInFile, 625
  SetOutFile, 625
OEChem::OESingleBondCount, 1000
OEChem::OESLNFlag, 857
OEChem::OESLNFlag::Canonical, 857
OEChem::OESLNFlag::Coords2D, 858
OEChem::OESLNFlag::Coords3D, 858
OEChem::OESLNFlag::DEFAULT, 858
OEChem::OESLNFlag::FCharge, 858
OEChem::OESLNFlag::Hydrogens, 857
OEChem::OESLNFlag::Isotopes, 857
OEChem::OESLNFlag::Kekule, 858
OEChem::OESLNFlag::Name, 858
OEChem::OESMARTSAtomFlag, 668
OEChem::OESMARTSAtomFlag::AAmap, 668
OEChem::OESMARTSAtomFlag::AnyAtomType, 668
OEChem::OESMARTSAtomFlag::AtomAll, 670
OEChem::OESMARTSAtomFlag::AtomDefault, 670
OEChem::OESMARTSAtomFlag::AtomExact, 671
OEChem::OESMARTSAtomFlag::AtomType, 668
OEChem::OESMARTSAtomFlag::Charge, 669
OEChem::OESMARTSAtomFlag::Charge0, 669
OEChem::OESMARTSAtomFlag::Chirality, 669
OEChem::OESMARTSAtomFlag::Connectivity, 669
OEChem::OESMARTSAtomFlag::DEFAULT, 670
OEChem::OESMARTSAtomFlag::DegreeCount, 669
OEChem::OESMARTSAtomFlag::HCount, 669
OEChem::OESMARTSAtomFlag::HCount0, 669
OEChem::OESMARTSAtomFlag::Hybridization, 669
OEChem::OESMARTSAtomFlag::IgnoreArom, 668
OEChem::OESMARTSAtomFlag::ImplicitHCount, 670
OEChem::OESMARTSAtomFlag::ImplicitHCount0, 670
OEChem::OESMARTSAtomFlag::Isotope, 670
OEChem::OESMARTSAtomFlag::LeavingGroup, 670
OEChem::OESMARTSAtomFlag::RingBondCount, 670
OEChem::OESMARTSAtomFlag::Valence, 670
OEChem::OESMARTSBondFlag, 671
OEChem::OESMARTSBondFlag::AnyBondType, 671
OEChem::OESMARTSBondFlag::AromBondType, 671
OEChem::OESMARTSBondFlag::BondAll, 672
OEChem::OESMARTSBondFlag::BondDefault, 672
OEChem::OESMARTSBondFlag::BondExact, 672
OEChem::OESMARTSBondFlag::BondTopo, 671
OEChem::OESMARTSBondFlag::BondType, 671
OEChem::OESMARTSBondFlag::Chirality, 671
OEChem::OESMARTSBondFlag::DEFAULT, 672
OEChem::OESmartsLexReplace, 1000
OEChem::OESmartsParseOpts, 858
OEChem::OESmartsParseOpts::Default, 858
OEChem::OESmartsParseOpts::None, 859
OEChem::OESmartsParseOpts::Optimize, 859
OEChem::OESmartsParseOpts::RingConstraint, 859
OEChem::OESmilesAtomCount, 1000
OEChem::OESMILESFlag, 859
OEChem::OESMILESFlag::AllBonds, 859
OEChem::OESMILESFlag::AtomMaps, 859
OEChem::OESMILESFlag::AtomStereo, 860
OEChem::OESMILESFlag::BondStereo, 860
OEChem::OESMILESFlag::Canonical, 860
OEChem::OESMILESFlag::DEFAULT, 861
OEChem::OESMILESFlag::EnhStereo, 860
OEChem::OESMILESFlag::ExtBonds, 860
OEChem::OESMILESFlag::Hydrogens, 860
OEChem::OESMILESFlag::ImpHCount, 860
OEChem::OESMILESFlag::ISOMERIC, 861
OEChem::OESMILESFlag::Isotopes, 860
OEChem::OESMILESFlag::Kekule, 861
OEChem::OESMILESFlag::RGroups, 861
OEChem::OESMILESFlag::SuperAtoms, 861
OEChem::OESmilesToMol, 1001
OEChem::OESortConfsBySDTag, 1001
OEChem::OESortConfsByTag, 1002
OEChem::OEString2OEAtomExpr, 1002

OEChem::OEString2OEBondExpr, 1002
OEChem::OEStringToMonomerType, 1056
OEChem::OEStringToPolymerType, 1056
OEChem::OEStyleGridType, 861
OEChem::OEStyleGridType::BEGIN, 861
OEChem::OEStyleGridType::Electrostatic, 861
OEChem::OEStyleGridType::END, 861
OEChem::OESubSearch, 625
AddConstraint, 626
AtomMatch, 627
ClearConstraints, 627
Constructors, 625
GetMaxMatches, 627
GetPattern, 627
Init, 627
Match, 628
operator bool, 626
SetMaxMatches, 628
SingleMatch, 629
OEChem::OESubSearchDatabase, 629
Constructors, 631
GetDatabaseType, 631
GetMatchIndices, 631
GetMatchTitles, 631
GetMolecule, 631
GetScreenType, 632
GetTitle, 632
IsValid, 632
NumMatches, 632
NumMolecules, 632
NumProcessors, 632
Open, 633
Search, 633
OEChem::OESubSearchDatabaseType, 862
OEChem::OESubSearchDatabaseType::Default, 862
OEChem::OESubSearchDatabaseType::InMemoryMultiThreaded, 862
OEChem::OESubSearchDatabaseType::MolDatabaseMultiThreaded, 862
OEChem::OESubSearchQuery, 633
Constructors, 634
GetMaxMatches, 635
SetFilter, 634
SetMaxMatches, 635
OEChem::OESubSearchResult, 635
Cancel, 636
Constructors, 636
GetMatchIndices, 636
GetSearchStatus, 636
IsValid, 636
NumMatches, 636
NumScreened, 637
NumSearched, 637
NumTargets, 637
NumTotalMatches, 637
OEChem::OESubSearchScreen, 637
Constructors, 638
GetScreenType, 638
IsValid, 639
OEChem::OESubSearchScreenType, 862
OEChem::OESubSearchScreenType::Default, 863
OEChem::OESubSearchScreenType::MDL, 863
OEChem::OESubSearchScreenType::Molecule, 863
OEChem::OESubSearchScreenType::SMARTS, 864
OEChem::OESubSearchScreenTypeBase, 639
GetName, 639
GetScreenType, 639
GetScreenTypeString, 639
GetSize, 640
GetVersion, 640
GetVersionString, 640
OEChem::OESubSearchStatus, 864
OEChem::OESubSearchStatus::Canceled, 864
OEChem::OESubSearchStatus::Finished, 864
OEChem::OESubSearchStatus::InProgress, 865
OEChem::OESubSearchStatus::Uninitialized, 865
OEChem::OESubSearchStatusToName, 1002
OEChem::OESubsetMol, 1003
OEChem::OESuppressHydrogens, 1006
OEChem::OESurfaceColorScheme, 865
OEChem::OESurfaceColorScheme::AtomColor, 865
OEChem::OESurfaceColorScheme::BEGIN, 865
OEChem::OESurfaceColorScheme::Concavity, 865
OEChem::OESurfaceColorScheme::Curvature, 865
OEChem::OESurfaceColorScheme::Distance, 865
OEChem::OESurfaceColorScheme::Electrostatic, 866
OEChem::OESurfaceColorScheme::END, 865
OEChem::OESurfaceColorScheme::GenericData, 866
OEChem::OESurfaceColorScheme::Grid, 866
OEChem::OESurfaceColorScheme::HBondPotential, 866
OEChem::OESurfaceColorScheme::Hydrophobicity, 866
OEChem::OESurfaceColorScheme::Solid, 866
OEChem::OESurfaceColorScheme::SurfacePotential, 866
OEChem::OESurfaceStyle, 866
OEChem::OESurfaceStyle::BEGIN, 866
OEChem::OESurfaceStyle::END, 866
OEChem::OESurfaceStyle::Mesh, 866
OEChem::OESurfaceStyle::Points, 867
OEChem::OESurfaceStyle::Solid, 868
OEChem::OESurfaceType, 869
OEChem::OESurfaceType::Accessible, 869
OEChem::OESurfaceType::BEGIN, 869
OEChem::OESurfaceType::END, 869

OEChem::OESurfaceType::Molecular, 869
OEChem::OESurfaceType::Off, 869
OEChem::OESurfaceType::VDW, 870
OEChem::OESweepMDLStereoGroups, 1006
OEChem::OESweepRotorCompressHydrogens, 1007
OEChem::OESweepStyles, 1007
OEChem::OESymmetryNumber, 1007
OEChem::OETerminalOCount, 1008
OEChem::OETerminalSCount, 1008
OEChem::OETheFunctionFormerlyKnownAsStripSalts, 1008
OEChem::OETokenizerBase, 640
    GetNextToken, 640
    ParseTitle, 640
OEChem::OETorsion, 641
    Constructors, 641
OEChem::OETrans, 641
    Clear, 642
    Compress, 642
    Constructors, 641
    Extract, 642
    GetInverse, 642
    GetTransforms, 642
    operator bool, 642
    PushBack, 642
    PushFront, 642
    Transform, 642
OEChem::OETransBase, 643
    CreateCopy, 643
    Extract, 643
    GetInverse, 643
    Transform, 643
OEChem::OETranslate, 1008
OEChem::OETranslation, 644
    Constructors, 644
    CreateCopy, 644
    Extract, 644
    GetInverse, 644
    GetTranslation, 645
OEChem::OETrimConformers, 1009
OEChem::OETripleBondCount, 1009
OEChem::OETriposAtomNames, 1009
OEChem::OETriposAtomType, 1010
OEChem::OETriposAtomTypeNames, 1009
OEChem::OETriposAtomTypes, 1010
OEChem::OETriposBondTypeNames, 1010
OEChem::OETriposOrderAtoms, 1010
OEChem::OETriposResonantTypeNames, 1010
OEChem::OETriposType, 870
OEChem::OETriposType::Al, 870
OEChem::OETriposType::Br, 870
OEChem::OETriposType::C1, 870
OEChem::OETriposType::C2, 870
OEChem::OETriposType::C3, 870
OEChem::OETriposType::Ca, 870
OEChem::OETriposType::Car, 870
OEChem::OETriposType::Ccat, 870
OEChem::OETriposType::Cl, 871
OEChem::OETriposType::Cooh, 871
OEChem::OETriposType::Croh, 871
OEChem::OETriposType::Crth, 871
OEChem::OETriposType::Cu, 871
OEChem::OETriposType::Du, 870
OEChem::OETriposType::F, 871
OEChem::OETriposType::Fe, 871
OEChem::OETriposType::H, 871
OEChem::OETriposType::I, 871
OEChem::OETriposType::K, 871
OEChem::OETriposType::Li, 872
OEChem::OETriposType::Lp, 872
OEChem::OETriposType::MAXTYPE, 874
OEChem::OETriposType::Mg, 872
OEChem::OETriposType::Mn, 872
OEChem::OETriposType::Mo, 872
OEChem::OETriposType::N1, 872
OEChem::OETriposType::N2, 872
OEChem::OETriposType::N3, 872
OEChem::OETriposType::N4, 872
OEChem::OETriposType::Na, 873
OEChem::OETriposType::Nam, 872
OEChem::OETriposType::Nar, 873
OEChem::OETriposType::Npl3, 873
OEChem::OETriposType::O2, 873
OEChem::OETriposType::O3, 873
OEChem::OETriposType::Oco2, 873
OEChem::OETriposType::P3, 873
OEChem::OETriposType::S2, 873
OEChem::OETriposType::S3, 873
OEChem::OETriposType::Se, 873
OEChem::OETriposType::Si, 874
OEChem::OETriposType::Sn, 874
OEChem::OETriposType::So, 874
OEChem::OETriposType::So2, 874
OEChem::OETriposType::Zn, 874
OEChem::OETriposTypeElement, 1011
OEChem::OETriposTypeIndex, 1011
OEChem::OETriposTypeName, 1011
OEChem::OETriposTypeNames, 1011
OEChem::OEUnaryToBinaryAnd, 645
    Constructors, 645
    CreateCopy, 645
    operator(), 645
OEChem::OEUnaryToBinaryOr, 645
    Constructors, 646
    CreateCopy, 646
    operator(), 646
OEChem::OEUnAtomPredParameter, 646
    Constructors, 647

OEChem::OEUnBondPredParameter, 647
    Constructors, 648
OEChem::OEUncolorMol, 1011
OEChem::OEUncolorStrategy, 874
OEChem::OEUncolorStrategy::BemisMurcko, 875
OEChem::OEUncolorStrategy::ConvertAtomTypeToC, 875
OEChem::OEUncolorStrategy::ConvertBondTypeToSingle, 875
OEChem::OEUncolorStrategy::Default, 874
OEChem::OEUncolorStrategy::RemoveAtomProperties, 875
OEChem::OEUncolorStrategy::RemoveAtomStereo, 875
OEChem::OEUncolorStrategy::RemoveBondStereo, 875
OEChem::OEUncolorStrategy::RemoveDimension, 875
OEChem::OEUncolorStrategy::RemoveGroupStereo, 875
OEChem::OEUncolorStrategy::UncolorNone, 874
OEChem::OEUniMolecularRxn, 648
    Constructors, 649
    GetClearCoordinates, 650
    GetOptions, 650
    GetValidateKekule, 651
    Init, 651
    operator bool, 650
    operator(), 650
    SetClearCoordinates, 652
    SetValidateKekule, 652
OEChem::OEUniMolecularRxnFixValence, 875
OEChem::OEUniMolecularRxnFixValence::All, 875
OEChem::OEUniMolecularRxnFixValence::Altered, 876
OEChem::OEUniMolecularRxnFixValence::DEFAULT, 876
OEChem::OEUniMolecularRxnFixValence::Default, 876
OEChem::OEUniMolecularRxnFixValence::Explicit, 876
OEChem::OEUniMolecularRxnOptions, 652
    Constructors, 653
    GetAlteredTag, 653
    GetClearCoordinates, 653
    GetFixValence, 654
    GetHydrogenConversions, 654
    GetMaxMatches, 654
    GetStrictSmirks, 654
    GetValidateKekule, 655
    SetAlteredTag, 655
    SetClearCoordinates, 655
    SetFixValence, 655
    SetHydrogenConversions, 656
    SetMaxMatches, 656
    SetStrictSmirks, 656
    SetValidateKekule, 656
OEChem::OEUpdateMMCIFData, 1012
OEChem::OEVectorBindings, 657
    Add, 657
    Constructors, 657
    Get, 657
OEChem::OEWeightedRMSD, 1012
OEChem::OEWrite2DRingDictionary, 1014
OEChem::OEWriteCDXFile, 1014
OEChem::OEWriteCIFFile, 1015
OEChem::OEWriteConstMolecule, 1014
OEChem::OEWriteCSVFile, 1015
OEChem::OEWriteCSVHeader, 1015
OEChem::OEWriteFASTAFile, 1016
OEChem::OEWriteMacroModelFile, 1016
OEChem::OEWriteMDLFile, 1016
OEChem::OEWriteMol2File, 1016
OEChem::OEWriteMolecule, 1016
OEChem::OEWriteMolReturnCode, 876
OEChem::OEWriteMolReturnCode::FormatNotWritable, 876
OEChem::OEWriteMolReturnCode::InvalidSDData, 877
OEChem::OEWriteMolReturnCode::MolInvalidForFileFormatSpec, 877
OEChem::OEWriteMolReturnCode::NoConformer, 876
OEChem::OEWriteMolReturnCode::OEBWriteFailure, 876
OEChem::OEWriteMolReturnCode::Success, 876
OEChem::OEWriteMolReturnCode::TooManyAtoms, 876
OEChem::OEWriteMolToBytes, 1017
OEChem::OEWriteMonomerSet, 1057
OEChem::OEWriteMOPACInputFile, 1018
OEChem::OEWritePDBFile, 1018
OEChem::OEWriteXYZFile, 1018
OEChem::OEZap7Version, 877
OEChem::OEZap7Version::CarbonylOxygen, 877
OEChem::OEZap7Version::Default, 877
OEChem::OEZap7Version::DoublyBondedOxygen, 877
OEChem::OEZap7Version::Undefined, 877
OEChem::OEZap9Version, 877
OEChem::OEZap9Version::Default, 878
OEChem::OEZap9Version::JCAMD2010, 878
OEChem::OEZap9Version::JMC2008, 878
OEChem::OEZap9Version::Undefined, 877
oecheminfo.cs
    Example Code, 317
OECombineMolComplexFragments, 222
OEConfIdxSelected, 488

OEDataFlow::OEMakeMatchField, 1263
OEDataFlow::OEMolRecord, 586
OEDataFlow::OEReadRecord, 1263
OEDataFlow::OERecord, 1261
OEDataFlow::OERecordReturnCode, 1192
OEDataFlow::OERecordReturnCode::MetadataMismatch, 1193
OEDataFlow::OERecordReturnCode::NoValue, 1193
OEDataFlow::OERecordReturnCode::NullValue, 1193
OEDataFlow::OERecordReturnCode::OK, 1192
OEDataFlow::OERecordReturnCode::TypeMismatch, 1193
OEDataFlow::Types, 1261
OEGetMolComplexComponents, 222
OEGetMolComplexFragments, 222
OEGetResidueAtoms, 218
OEHasAtomIdx, 484
OEHasBondIdx, 484
OEHasConfIdx, 484
OEHierChain, 214
OEHierFragment, 214
OEHierResidue, 214
OEHierView, 214
OEIsStandardProteinResidue, 218
OEKekulize, 135
OELibraryGen, 117, 121
OEMatchAtom, 533
OEMatchGraphMol, 533
OEMatchMolBase, 533
OEMath::OEBox, 1301
  Constructors, 1301
  Extend, 1301
  GetArea, 1301
  GetVolume, 1301
  GetXDim, 1301
  GetXMax, 1302
  GetXMid, 1302
  GetXMin, 1302
  GetYDim, 1301
  GetYMax, 1302
  GetYMid, 1302
  GetYMin, 1303
  GetZDim, 1302
  GetZMax, 1302
  GetZMid, 1302
  GetZMin, 1303
  IsInside, 1303
  Setup, 1303
OEMath::OEBoxArea, 1310
OEMath::OEBoxExtend, 1310
OEMath::OEBoxTranslate, 1310
OEMath::OEBoxVolume, 1311
OEMath::OEBoxXDim, 1311
OEMath::OEBoxXMid, 1311
OEMath::OEBoxYDim, 1311
OEMath::OEBoxYMid, 1311
OEMath::OEBoxZDim, 1311
OEMath::OEBoxZMid, 1311
OEMath::OECenter, 1311
OEMath::OEConst, 1309
OEMath::OEConst::c, 1309
OEMath::OEConst::cal2J, 1310
OEMath::OEConst::e, 1309
OEMath::OEConst::eps0, 1310
OEMath::OEConst::h, 1309
OEMath::OEConst::hbar, 1309
OEMath::OEConst::k, 1309
OEMath::OEConst::me, 1310
OEMath::OEConst::mp, 1310
OEMath::OEConst::mu, 1310
OEMath::OEConst::NA, 1309
OEMath::OEConst::R, 1309
OEMath::OEGeom2DAdd, 1312
OEMath::OEGeom2DAngle, 1312
OEMath::OEGeom2DCopy, 1312
OEMath::OEGeom2DDistance, 1312
OEMath::OEGeom2DDistance2, 1312
OEMath::OEGeom2DDotProd, 1313
OEMath::OEGeom2DMagnitude, 1313
OEMath::OEGeom2DMidpoint, 1313
OEMath::OEGeom2DNormalize, 1313
OEMath::OEGeom2DRotate, 1313
OEMath::OEGeom2DScale, 1314
OEMath::OEGeom2DSetLength, 1314
OEMath::OEGeom2DSubtract, 1314
OEMath::OEGeom2DTranslate, 1314
OEMath::OEGeom3DAbsTorsion, 1314
OEMath::OEGeom3DAdd, 1314
OEMath::OEGeom3DAllZero, 1315
OEMath::OEGeom3DAngle, 1315
OEMath::OEGeom3DAngleCoord, 1315
OEMath::OEGeom3DCopy, 1315
OEMath::OEGeom3DCrossProd, 1316
OEMath::OEGeom3DDistance, 1316
OEMath::OEGeom3DDistance2, 1316
OEMath::OEGeom3DDotProd, 1316
OEMath::OEGeom3DEulerRotate, 1316
OEMath::OEGeom3DEulerToQuaternion, 1316
OEMath::OEGeom3DEulerToRotMatrix, 1316
OEMath::OEGeom3DGetCenterAndExtents, 1317
OEMath::OEGeom3DGetCenterOfMass, 1317
OEMath::OEGeom3DMagnitude, 1317
OEMath::OEGeom3DMakeQuaternion, 1317
OEMath::OEGeom3DMatrixIdentity, 1318
OEMath::OEGeom3DMatrixInvert, 1318
OEMath::OEGeom3DMatrixMultiply, 1318
OEMath::OEGeom3DMatrixTranspose, 1318

OEMath::OEGeom3DNormalize, 1318
OEMath::OEGeom3DNormalizeQuaternion, 1318
OEMath::OEGeom3DPlanarCoord, 1319
OEMath::OEGeom3DQuaternionIdentity, 1319
OEMath::OEGeom3DQuaternionInvert, 1319
OEMath::OEGeom3DQuaternionRotate, 1319
OEMath::OEGeom3DQuaternionToRotMatrix, 1320
OEMath::OEGeom3DReflect, 1320
OEMath::OEGeom3DReflectCoord, 1320
OEMath::OEGeom3DRotate, 1320
OEMath::OEGeom3DRotVectorToTransform, 1320
OEMath::OEGeom3DScale, 1321
OEMath::OEGeom3DSetLength, 1321
OEMath::OEGeom3DSubtract, 1321
OEMath::OEGeom3DTetraCoord, 1321
OEMath::OEGeom3DTorsion, 1322
OEMath::OEGeom3DTranslate, 1322
OEMath::OEGeom3DUnitQuaternionRotate, 1322
OEMath::OEGeom3DVolume, 1323
OEMath::OEGeomDotProd, 1323
OEMath::OEGeomNormalizeQuaternion, 1323
OEMath::OEGeomQuaternionMultiply, 1319, 1323
OEMath::OEInBox, 1323
OEMath::OEInSphere, 1323
OEMath::OEKernelPLS, 1303
    Constructor, 1304
    Fit, 1304
    GetB0, 1304
    GetBValues, 1304
    GetNumFeaturesUsed, 1304
    Predict, 1304
OEMath::OEMath, 1308
OEMath::OEMath::Deg2Rad, 1309
OEMath::OEMath::Pi, 1309
OEMath::OEMath::Pi2, 1309
OEMath::OEMath::Rad2Deg, 1308
OEMath::OEMatrix, 1305
    AddRow, 1305
    Clear, 1305
    Constructors, 1305
    GetData, 1305
    GetNumColumns, 1306
    GetNumRows, 1306
    Resize, 1306
    SetRowData, 1306
    Transpose, 1306
OEMath::OESetupBox, 1323
OEMath::OESetupBoxCenterAndExtents, 1324
OEMath::OESphere, 1306
    Constructor, 1306
    GetArea, 1306
    GetRad, 1307
    GetVolume, 1307
    GetX, 1307
    GetY, 1307
    GetZ, 1307
    IsInside, 1307
    SetCenter, 1307
    SetRad, 1307
OEMath::OESphereArea, 1324
OEMath::OESphereVolume, 1324
OEMath::OESquareMatrix, 1308
    Constructors, 1308
    Resize, 1308
OEMolBase, 50
OEMolDatabase, 145
oemolistream, 13
oemolostream, 13
oemolstreambase, 13
OEPartPredAtom, 601
OEPDBClassify, 218
OEPlatform::OEAddLicenseData, 1292
OEPlatform::OEAddLicenseFromHttp, 1292
OEPlatform::OEAddLicenseKey, 1292
OEPlatform::OEBigEndian, 1292
OEPlatform::OECloseIStream, 1292
OEPlatform::OECloseOStream, 1292
OEPlatform::OECompress, 1293
OEPlatform::OECompressionAvailable, 1293
OEPlatform::OECondition, 1264
    Acquire, 1264
    Broadcast, 1264
    Constructors, 1264
    Release, 1264
    Signal, 1264
    Wait, 1264
OEPlatform::OECreateDirectory, 1293
OEPlatform::OECurrentTime, 1293
OEPlatform::OEDirectoryScan, 1264
    Close, 1265
    Constructors, 1264
    Next, 1265
    operator bool, 1265
OEPlatform::oeendl, 1293
OEPlatform::oeends, 1293
OEPlatform::OEFile, 1265
    AbsPath, 1265
    Constructors, 1265
    DirContents, 1265
    DirPath, 1265
    Exists, 1266
    FileName, 1266
    IsDirectory, 1266
    IsExecutable, 1266
    IsFile, 1266
    IsLink, 1266
    IsReadable, 1266
    IsWritable, 1266

Refresh, 1266
Set, 1266
Size, 1266
OEPlatform::OEFileCorrectSeparators, 1293
OEPlatform::OEFileCreate, 1294
OEPlatform::OEFileDelete, 1294
OEPlatform::OEFileDetermineName, 1294
OEPlatform::OEFileDeterminePath, 1294
OEPlatform::OEFileDeterminePathAndName, 1294
OEPlatform::OEFileExists, 1294
OEPlatform::OEFileIsDirectory, 1294
OEPlatform::OEFileIsFile, 1294
OEPlatform::OEFileIsLink, 1294
OEPlatform::OEFileIsReadable, 1294
OEPlatform::OEFileIsWritable, 1294
OEPlatform::OEFileRandomName, 1295
OEPlatform::OEFileSize, 1295
OEPlatform::OEFileTempDescriptor, 1295
OEPlatform::OEFileTempDirectory, 1295
OEPlatform::OEFileTempPath, 1295
OEPlatform::oeflush, 1293
OEPlatform::OEGetAbsolutePath, 1295
OEPlatform::OEGetAllocaThreshold, 1296
OEPlatform::OEGetCurrentWorkingDirectory, 1296
OEPlatform::OEGetDomainName, 1296
OEPlatform::OEGetHostIdent, 1296
OEPlatform::OEGetHostName, 1296
OEPlatform::OEGetIPAddress, 1296
OEPlatform::OEGetMachineName, 1296
OEPlatform::OEGetNumProcessors, 1296
OEPlatform::OEGetPlatformInfo, 1297
OEPlatform::OEGetProcessMemUsage, 1297
OEPlatform::OEGetProcessorInfo, 1297
OEPlatform::OEGetTimeOfDay, 1297
OEPlatform::OEGetUserAppDataDir, 1297
OEPlatform::OEGetUserDocumentsDir, 1297
OEPlatform::OEGetUserHomeDir, 1297
OEPlatform::OEGetUserName, 1297
OEPlatform::OEHalfFloat, 1267
Constructors, 1267
GetValue, 1267
OEPlatform::oeifstream, 1274
Constructors, 1274
fd, 1274
openfd, 1274
pread, 1275
OEPlatform::oeigzstream, 1275
Constructors, 1275
OEPlatform::oeiofstream, 1276
Constructors, 1276
pread, 1276
OEPlatform::OEIsNaN, 1297
OEPlatform::OEIsNormalNum, 1298
OEPlatform::oeisstream, 1276
clear, 1277
Constructors, 1277
getbuffer, 1277
reset, 1277
set, 1277
OEPlatform::oeistdstream, 1278
clear, 1278
Constructors, 1278
set, 1278
stream, 1279
OEPlatform::oeistream, 1279
close, 1279
Constructors, 1279
get, 1279
getbyte, 1280
getline, 1280
gettoken, 1280
open, 1280
peekbyte, 1280
read, 1281
skip, 1281
skipbyte, 1281
skipline, 1281
OEPlatform::oeiwrapperstream, 1281
clear, 1282
Constructors, 1282
set, 1282
stream, 1282
OEPlatform::OELocalTime, 1298
OEPlatform::OELock, 1267
OEPlatform::OELockCondition, 1268
Constructors, 1268
OEPlatform::OELockSpinlock, 1268
OEPlatform::oemultiostream, 1282
addstream, 1283
clear, 1283
Constructors, 1283
OEPlatform::OEMutex, 1268
Acquire, 1269
Constructors, 1269
Release, 1269
OEPlatform::oeofstream, 1283
append, 1284
Constructors, 1284
fd, 1284
openfd, 1284
OEPlatform::oeogzstream, 1284
Constructors, 1285
OEPlatform::OEOnce, 1269
GetValue, 1270
OEPlatform::OEOpenIStream, 1298
OEPlatform::OEOpenOStream, 1298
OEPlatform::oeosstream, 1285

clear, 1286
Constructors, 1285
set, 1286
str, 1286
OEPlatform::oeostdstream, 1286
clear, 1286
Constructors, 1286
set, 1287
stream, 1287
OEPlatform::oeostream, 1287
close, 1287
Constructors, 1287
flush, 1287
open, 1288
putbyte, 1288
write, 1288
OEPlatform::oeowrapperstream, 1288
clear, 1289
Constructors, 1289
set, 1289
stream, 1289
OEPlatform::OEPlatform, 1291
OEPlatform::OEPlatform::oeerr, 1291
OEPlatform::OEPlatform::oein, 1291
OEPlatform::OEPlatform::oenul, 1291
OEPlatform::OEPlatform::oeout, 1291
OEPlatform::OEProcessFileName, 1299
OEPlatform::OEProcessSystemCommand, 1299
OEPlatform::OERegisterIStream, 1299
OEPlatform::OERegisterOStream, 1299
OEPlatform::OERenameFile, 1299
OEPlatform::OESetAllocaThreshold, 1299
OEPlatform::OESetLicenseFile, 1300
OEPlatform::OESleep, 1300
OEPlatform::OESpinlock, 1270
Acquire, 1271
Constructors, 1271
Release, 1271
OEPlatform::oestream, 1289
Constructors, 1290
eof, 1290
length, 1290
operator bool, 1290
rewind, 1290
seek, 1290
size, 1291
tell, 1291
OEPlatform::OEThread, 1271
Constructors, 1271
GetArg, 1271
GetReturn, 1271
GetStackSize, 1272
Join, 1272
Run, 1272
SetArg, 1272
SetReturn, 1272
SetStackSize, 1272
Start, 1272
OEPlatform::OEThreadLocal, 1273
Constructors, 1273
OEPlatform::OEToolkitsGetRelease, 1300
OEPlatform::OETryMutex, 1273
Acquire, 1273
Constructors, 1273
Release, 1273
Try, 1273
OEPlatform::OEUncompress, 1300
OEPlatform::OEUnlink, 1300
OEPlatform::OEUnlinkDirectory, 1300
OEReadMDLQueryFile, 105
OEReadMDLReactionQueryFile, 132
OEResidue, 211
OEResidues, 214
OESequenceAlignment, 216
OESplitMolComplex, 222
OESwapAIEResidueAtoms, 227
OESystem::IsFalse, 1057
OESystem::IsTrue, 1058
OESystem::OEAddContour, 1200
OESystem::OEAddGrid, 1200
OESystem::OEAnd, 1058
Constructors, 1059
CreateCopy, 1059
operator(), 1059
OESystem::OEAnnotateArrow, 1200
OESystem::OEAnnotateBox, 1200
OESystem::OEAnnotateCylinder, 1201
OESystem::OEAnnotateEllipsoid, 1201
OESystem::OEAnnotateLine, 1201
OESystem::OEAnnotateSphere, 1201
OESystem::OEAnnotateSpline, 1201
OESystem::OEAnnotateSplineSurface, 1201
OESystem::OEAnnotateSymbol, 1201
OESystem::OEAnnotateText, 1201
OESystem::OEAnnotateTorus, 1202
OESystem::OEAnnotation, 1059
AddChild, 1060
Constructors, 1060
CreateCopy, 1060
GetChildren, 1060
GetColor, 1060
GetDataType, 1060
GetRenderHint, 1060
GetTitle, 1060
GetType, 1060
IsDataType, 1061
SetColor, 1061
SetRenderHint, 1061

SetTitle, 1061
SetType, 1061
OESystem::OEAnnotationPrimitive, 1187
OESystem::OEAnnotationPrimitive::Arrow, 1187
OESystem::OEAnnotationPrimitive::Box, 1187
OESystem::OEAnnotationPrimitive::Container, 1187
OESystem::OEAnnotationPrimitive::Cylinder, 1187
OESystem::OEAnnotationPrimitive::Ellipsoid, 1187
OESystem::OEAnnotationPrimitive::Line, 1187
OESystem::OEAnnotationPrimitive::Sphere, 1187
OESystem::OEAnnotationPrimitive::Spline, 1187
OESystem::OEAnnotationPrimitive::SplineSurface, 1187
OESystem::OEAnnotationPrimitive::Surface, 1187
OESystem::OEAnnotationPrimitive::Symbol, 1188
OESystem::OEAnnotationPrimitive::Text, 1187
OESystem::OEAnnotationPrimitive::Torus, 1187
OESystem::OEAnnotationRenderHint, 1188
OESystem::OEAnnotationRenderHint::Mesh, 1188
OESystem::OEAnnotationRenderHint::Points, 1188
OESystem::OEAnnotationRenderHint::Solid, 1188
OESystem::OEAttachData, 1202
OESystem::OEBase, 1061
AddBaseData, 1063
AddData, 1063
Clear, 1063
Constructors, 1062
CreateCopy, 1063
DeleteData, 1063
GetBoolData, 1064
GetData, 1064
GetDataIter, 1064
GetDataType, 1064
GetDoubleData, 1064
GetFloatData, 1065
GetIntData, 1065
GetStringData, 1065
HasData, 1065
IsDataType, 1065
operator+=, 1062
SetBaseData, 1065
SetBoolData, 1066
SetData, 1066
SetDoubleData, 1066
SetFloatData, 1066
SetIntData, 1067
SetStringData, 1067
OESystem::OEBaseData, 1067
Constructors, 1067
CreateCopy, 1068
GetDataType, 1068
GetSize, 1068
GetTag, 1068
OESystem::OEBaseDataT, 1068
Constructors, 1069
CreateCopy, 1069
GetData, 1069
GetDataType, 1069
GetSize, 1069
SetData, 1069
OESystem::OEBaseDataTPtr, 1069
OESystem::OEBDataLengthBytes, 1202
OESystem::OEBFloatEndian, 1202
OESystem::OEBFPosTEndian, 1202
OESystem::OEBinaryAnd, 1070
Constructors, 1070
CreateCopy, 1071
operator(), 1070
OESystem::OEBinaryFalse, 1071
CreateCopy, 1071
operator(), 1071
OESystem::OEBinaryFunction, 1071
CreateCopy, 1072
operator(), 1072
OESystem::OEBinaryIOHandler, 1072
Clear, 1072
Constructors, 1072
CreateCopy, 1072
GetBaseWriteFxns, 1072
GetDataType, 1073
GetDefaultReadFxn, 1073
GetDefaultWriteFxn, 1073
GetReadFxns, 1073
GetWriteFxns, 1073
IsDataType, 1073
ReplaceReadFxn, 1073
ReplaceWriteFxn, 1073
SetBaseWriteFxn, 1073
SetReadFxn, 1073
SetWriteFxn, 1073
OESystem::OEBinaryIOHandlerBase, 1074
Clear, 1074
GetBaseWriteFxns, 1074
GetDefaultReadFxn, 1074
GetDefaultWriteFxn, 1074
GetReadFxns, 1074
GetWriteFxns, 1074
ReplaceReadFxn, 1074
ReplaceWriteFxn, 1075
SetBaseWriteFxn, 1075
SetReadFxn, 1075
SetWriteFxn, 1075
OESystem::OEBinaryNot, 1075

Constructors, 1075
CreateCopy, 1076
operator(), 1075
OESystem::OEBinaryOr, 1076
Constructors, 1076
CreateCopy, 1076
operator(), 1076
OESystem::OEBinaryPredicate, 1076
CreateCopy, 1077
CreatePredicateCopy, 1077
operator(), 1077
OESystem::OEBinaryTrue, 1077
CreateCopy, 1077
operator(), 1077
OESystem::OEBInitializer, 1061
Constructors, 1061
OESystem::OEBIntEndian, 1202
OESystem::OEBitIsPowerOf2, 1204
OESystem::OEBitVector, 1078
ClearBits, 1078
Constructors, 1078
CountBits, 1078
CountRangeBits, 1078
FirstBit, 1079
FromHexString, 1079
GetData, 1079
GetSize, 1079
IsBitOn, 1079
IsEmpty, 1079
LastBit, 1080
NegateBits, 1080
NextBit, 1080
PrevBit, 1080
SetBitOff, 1080
SetBitOn, 1080
SetData, 1080
SetRangeOff, 1081
SetRangeOn, 1081
SetSize, 1081
ToggleBit, 1082
ToHexString, 1081
OESystem::OEBitVectorAnd, 1204
OESystem::OEBitVectorEqual, 1204
OESystem::OEBitVectorNot, 1204
OESystem::OEBitVectorNumBytes, 1205
OESystem::OEBitVectorNumWords, 1205
OESystem::OEBitVectorOr, 1205
OESystem::OEBitVectorSub, 1205
OESystem::OEBitVectorXor, 1205
OESystem::OEBoolParameter, 1244
Constructors, 1245
OESystem::OEBoundedBuffer, 1082
Constructors, 1082
Empty, 1082
Full, 1083
Get, 1083
Peek, 1083
Put, 1083
Size, 1083
OESystem::OEBPeekTag, 1202
OESystem::OEBReadData, 1203
OESystem::OEBReadDataLength, 1203
OESystem::OEBReadOEBase, 1203
OESystem::OEBReadPOD, 1203
OESystem::OEBReadTag, 1203
OESystem::OEBShortEndian, 1203
OESystem::OEBWriteData, 1203
OESystem::OEBWriteDataLength, 1204
OESystem::OEBWriteOEBase, 1204
OESystem::OEBWritePOD, 1204
OESystem::OEBWriteTag, 1204
OESystem::OECastData, 1205
OESystem::OECheckHelp, 1205
OESystem::OEClearGridBoundary, 1206
OESystem::OEColor, 1083
Constructors, 1085
CreateCopy, 1085
GetA, 1086
GetB, 1086
GetG, 1086
GetPackedColor, 1086
GetR, 1085
GetRGBA, 1086
GetText, 1086
Set, 1087
SetA, 1088
SetB, 1088
SetG, 1087
SetPackedColor, 1089
SetR, 1087
SetRGBA, 1089
SetText, 1089
OESystem::OEColorCreate, 1206
OESystem::OEColorGet, 1207
OESystem::OEColorGradientBase, 1089
AddStop, 1090
Constructors, 1090
DeleteStop, 1090
GetColorAt, 1090
GetMaxColor, 1090
GetMaxValue, 1091
GetMinColor, 1091
GetMinValue, 1091
GetStops, 1091
IsEmpty, 1091
NumStops, 1091
TransformValue, 1091
OESystem::OEColorPacked, 1207

OESystem::OEColorSet, 1207
OESystem::OEColorStop, 1091
    Constructors, 1092
    GetColor, 1092
    GetValue, 1092
OESystem::OEConcatIter, 1092
    Constructors, 1093
    Copy, 1093
    operator bool, 1093
    Push, 1093
    ToFirst, 1093
    ToLast, 1093
OESystem::OEConfigure, 1208
OESystem::OEConfigureItf, 1262
OESystem::OEConfigureOpts, 1262
OESystem::OEConsoleProgressTracer, 1093
    Aborted, 1094
    Constructors, 1093
    Pop, 1094
    Push, 1094
    Refresh, 1094
    SetProgress, 1094
    SetTask, 1094
OESystem::OEConstParameter, 1245
    AddLegalEntry, 1246
    Constructors, 1246
OESystem::OEContainer, 1094
    Constructors, 1095
    CreateCopy, 1095
    GetDataType, 1095
    IsDataType, 1095
OESystem::OECopyArray, 1211
OESystem::OECopyBaseData, 1212
OESystem::OECopyBits, 1212
OESystem::OECount, 1212
OESystem::OECycleTimer, 1095
    Constructors, 1095
    Elapsed, 1095
    Lap, 1096
    Start, 1096
OESystem::OEDivideGrid, 1212
OESystem::OEDots, 1096
    Constructors, 1096
    GetBigStep, 1097
    GetCounts, 1097
    GetSmallStep, 1097
    SetBigStep, 1097
    SetSmallStep, 1097
    Total, 1097
    Update, 1097
OESystem::OEDoubleParameter, 1246
    Constructors, 1247
OESystem::OEErrorHandler, 1098
    Clear, 1098
    Constructors, 1098
    Count, 1098
    Debug, 1098
    Error, 1099
    Fatal, 1099
    GetHandlerImpl, 1099
    GetLevel, 1099
    GetStrict, 1099
    Info, 1100
    Msg, 1100
    SetHandlerImpl, 1100
    SetLevel, 1100
    SetOutputStream, 1100
    SetStrict, 1100
    Usage, 1101
    Verbose, 1101
    Warning, 1101
OESystem::OEErrorHandlerImplBase, 1101
    Clear, 1101
    Constructors, 1101
    Count, 1102
    CreateCopy, 1102
    GetLevel, 1102
    GetStrict, 1102
    IncrementCount, 1102
    Msg, 1102
    SetLevel, 1102
    SetStrict, 1102
OESystem::OEErrorLevel, 1188
OESystem::OEErrorLevel::Debug, 1188
OESystem::OEErrorLevel::Default, 1189
OESystem::OEErrorLevel::Error, 1188
OESystem::OEErrorLevel::Fatal, 1188
OESystem::OEErrorLevel::Info, 1188
OESystem::OEErrorLevel::MAX, 1189
OESystem::OEErrorLevel::MIN, 1189
OESystem::OEErrorLevel::Quiet, 1188
OESystem::OEErrorLevel::Undefined, 1188
OESystem::OEErrorLevel::Usage, 1188
OESystem::OEErrorLevel::Verbose, 1188
OESystem::OEErrorLevel::Warning, 1188
OESystem::OEErrorLevelToString, 1212
OESystem::OEExponentColorGradient, 1102
    Constructors, 1103
    GetExponent, 1104
    SetExponent, 1104
    TransformValue, 1104
OESystem::OEExponentialColorGradient, 1104
    Constructors, 1105
    TransformValue, 1106
OESystem::OEFileExtension, 1213
OESystem::OEFileExtensions, 1213
OESystem::OEFixedGrid, 1106
    CreateCopy, 1107

GetDataType, 1108
GetValues, 1108
IsDataType, 1108
operator(), 1107
SetValues, 1108
OESystem::OEFixedMultiGrid, 1108
CreateCopy, 1109
GetDataType, 1109
GetNumGrids, 1109
GetValues, 1109
IsDataType, 1109
SetNumGrids, 1109
OESystem::OEFloatParameter, 1247
Constructors, 1248
OESystem::OEGaussian, 1109
Constructors, 1110
CreateCopy, 1110
GetCenter, 1110
GetColor, 1111
GetDataType, 1111
GetPrefactor, 1111
GetWidth, 1111
GetX, 1111
GetY, 1111
GetZ, 1111
IsDataType, 1111
SetCenter, 1111
SetColor, 1111
SetPrefactor, 1111
SetWidth, 1112
SetX, 1112
SetY, 1112
SetZ, 1112
OESystem::OEGaussianBase, 1112
GetCenter, 1112
GetColor, 1112
GetDataType, 1112
GetPrefactor, 1113
GetWidth, 1113
GetX, 1113
GetY, 1113
GetZ, 1113
IsDataType, 1113
SetCenter, 1113
SetColor, 1113
SetPrefactor, 1113
SetWidth, 1113
SetX, 1113
SetY, 1114
SetZ, 1114
OESystem::OEGaussianContainer, 1114
AddGaussian, 1114
Clear, 1114
Constructors, 1114
CreateCopy, 1115
DeleteGaussian, 1115
DeleteGaussians, 1115
GetColors, 1115
GetCoords, 1115
GetDataType, 1115
GetGaussians, 1115
GetPrefactors, 1115
GetWidths, 1115
IsDataType, 1115
NumGaussians, 1116
OESystem::OEGaussianContainerBase, 1116
AddGaussian, 1116
Clear, 1116
DeleteGaussian, 1116
DeleteGaussians, 1116
GetColors, 1116
GetCoords, 1116
GetDataType, 1117
GetGaussians, 1117
GetPrefactors, 1117
GetWidths, 1117
IsDataType, 1117
NumGaussians, 1117
OESystem::OEGetBitCounts, 1213
OESystem::OEGetCenterAndExtents, 1213
OESystem::OEGetColors, 1213
OESystem::OEGetContourColor, 1214
OESystem::OEGetContourLevel, 1214
OESystem::OEGetContrastColors, 1214
OESystem::OEGetDataType, 1215
OESystem::OEGetDeepColors, 1215
OESystem::OEGetFileEndian, 1215
OESystem::OEGetGridFileType, 1216
OESystem::OEGetGridFormatExtension, 1216
OESystem::OEGetGridFormatString, 1216
OESystem::OEGetHostEndian, 1217
OESystem::OEGetKappa, 1217
OESystem::OEGetLightColors, 1217
OESystem::OEGetMemPoolMode, 1217
OESystem::OEGetOEBReturnString, 1218
OESystem::OEGetPrefactor, 1218
OESystem::OEGetRadius, 1218
OESystem::OEGetTag, 1218
OESystem::OEGetThreadSafe, 1218
OESystem::OEGetVividColors, 1218
OESystem::OEGetWidth, 1219
OESystem::OEGrid, 1117
Clear, 1119
Constructors, 1118
CreateCopy, 1119
GetDataType, 1119
IsDataType, 1119
SetAll, 1119

SetDim, 1119
SetMid, 1119
SetSpacing, 1120
SetXDim, 1120
SetXMid, 1120
SetYDim, 1120
SetYMid, 1120
SetZDim, 1120
SetZMid, 1120
OESystem::OEGridAbstractBase, 1120
Clear, 1121
GetDataType, 1121
GetDim, 1121
GetMid, 1121
GetSize, 1121
GetSpacing, 1121
GetTitle, 1122
GetXDim, 1122
GetXMid, 1122
GetYDim, 1122
GetYMid, 1122
GetZDim, 1122
GetZMid, 1122
IsDataType, 1122
IsSpacingSet, 1122
SetTitle, 1122
OESystem::OEGridAvgVal, 1219
OESystem::OEGridBase, 1122
Clear, 1123
CreateCopy, 1123
ElementToGridIdx, 1123
ElementToSpatialCoord, 1124
GetDataType, 1124
GetDim, 1124
GetMid, 1124
GetSize, 1124
GetSpacing, 1124
GetTitle, 1124
GetX, 1124
GetXDim, 1125
GetXIdx, 1125
GetXInc, 1125
GetXMax, 1125
GetXMid, 1125
GetXMin, 1125
GetY, 1125
GetYDim, 1126
GetYIdx, 1126
GetYInc, 1126
GetYMax, 1126
GetYMid, 1126
GetYMin, 1126
GetZ, 1126
GetZDim, 1127
GetZIdx, 1127
GetZInc, 1127
GetZMax, 1127
GetZMid, 1127
GetZMin, 1127
GridIdxToElement, 1127
GridIdxToSpatialCoord, 1128
IsDataType, 1128
IsInGrid, 1128
IsSpacingSet, 1128
IsTitleSet, 1128
IsXMidSet, 1128
IsYMidSet, 1129
IsZMidSet, 1129
SetTitle, 1129
SpatialCoordToElement, 1129
SpatialCoordToGridIdx, 1129
OESystem::OEGridCopyContents, 1219
OESystem::OEGridCopyGeometry, 1219
OESystem::OEGridFileType, 1189
OESystem::OEGridFileType::Ascii, 1189
OESystem::OEGridFileType::CCP4, 1189
OESystem::OEGridFileType::Grasp, 1189
OESystem::OEGridFileType::GRD, 1189
OESystem::OEGridFileType::UNDEFINED, 1189
OESystem::OEGridFileType::XPLOR, 1189
OESystem::OEGridGetArch, 1219
OESystem::OEGridGetLicensee, 1220
OESystem::OEGridGetPlatform, 1220
OESystem::OEGridGetRelease, 1220
OESystem::OEGridGetSite, 1220
OESystem::OEGridGetVersion, 1220
OESystem::OEGridHasNonZeroValue, 1220
OESystem::OEGridIsLicensed, 1220
OESystem::OEGridMaskType, 1189
OESystem::OEGridMaskType::Default, 1190
OESystem::OEGridMaskType::Gaussian, 1190
OESystem::OEGridMaskType::GaussianMinus, 1190
OESystem::OEGridMaskType::GaussianSmooth, 1190
OESystem::OEGridMaskType::GaussianSmoothMinus, 1190
OESystem::OEGridMaskType::GridMinus, 1189
OESystem::OEGridMaskType::GridPlus, 1189
OESystem::OEGridMaskType::ValueMinus, 1189
OESystem::OEGridMaskType::ValuePlus, 1189
OESystem::OEGridMaxVal, 1221
OESystem::OEGridMinVal, 1221
OESystem::OEGridSameGeometry, 1221
OESystem::OEGridSetAllElements, 1222
OESystem::OEGridSizeMultiply, 1221
OESystem::OEGridType, 1190
OESystem::OEGridType::DifferenceMap, 1190
OESystem::OEGridType::ElectrostaticGrid, 1190

OESystem::OEGridType::EonGrid, 1190
OESystem::OEGridType::ETGrid, 1190
OESystem::OEGridType::FredGrid, 1190
OESystem::OEGridType::GenericGrid, 1190
OESystem::OEGridType::MaxGridType, 1190
OESystem::OEGridType::RegularMap, 1190
OESystem::OEHasRole, 1129
    Constructors, 1130
    CreateCopy, 1130
    operator(), 1130
OESystem::OEHasTag, 1222
OESystem::OEHeader, 1130
    AddHistory, 1131
    Constructors, 1131
    GetCommandLine, 1131
    GetComment, 1131
    GetDate, 1131
    GetHistory, 1131
    GetInterface, 1131
    GetOERelease, 1131
    GetOEVersion, 1132
    GetPlatform, 1132
    GetProgram, 1132
    GetVersion, 1132
    SetCommandLine, 1132
    SetComment, 1132
    SetDate, 1132
    SetInterface, 1132
    SetOERelease, 1132
    SetOEVersion, 1133
    SetPlatform, 1133
    SetProgram, 1133
    SetVersion, 1133
OESystem::OEInitAnnotationHandlers, 1222
OESystem::OEInitGridHandlers, 1222
OESystem::OEInitializeHandler, 1222
OESystem::OEInterface, 1133
    AddDetail, 1134
    AddInterface, 1134
    AddParameter, 1134
    Clear, 1134
    Constructors, 1133
    DeleteInterface, 1135
    DeleteParameter, 1135
    Factory, 1135
    Get, 1135
    GetBrief, 1135
    GetDetail, 1135
    GetInterface, 1135
    GetInterfaces, 1136
    GetList, 1136
    GetName, 1136
    GetOrderPriority, 1136
    GetParameter, 1136
    GetParameters, 1136
    GetTypedParameter, 1137
    Has, 1137
    Parent, 1137
    SetBrief, 1137
    SetName, 1137
    SetOrderPriority, 1137
OESystem::OEInterpolateBetweenGrids, 1222
OESystem::OEIntParameter, 1249
    Constructors, 1250
OESystem::OEIsEmpty, 1222
OESystem::OEIsFalse, 1138
    CreateCopy, 1138
    operator(), 1138
OESystem::OEIsGrid, 1223
OESystem::OEIsReadableGrid, 1223
OESystem::OEIsSafeCast, 1223
OESystem::OEIsSameTypeTester, 1223
OESystem::OEIsTrue, 1138
    CreateCopy, 1139
    operator(), 1139
OESystem::OEIsWriteableGrid, 1223
OESystem::OEIter, 1139
    Constructors, 1139
    Copy, 1140
    operator bool, 1140
    operator->, 1139
    Push, 1140
    Sort, 1140
    ToFirst, 1140
    ToLast, 1140
OESystem::OEIterBase, 1140
    Constructors, 1141
    Copy, 1141
    operator bool, 1141
    operator->, 1141
    Push, 1141
    ToFirst, 1141
    ToLast, 1141
OESystem::OELinearColorGradient, 1141
    Constructors, 1142
    TransformValue, 1143
OESystem::OELinearInterpolate, 1223
OESystem::OELogarithmicColorGradient, 1143
    Constructors, 1144
    TransformValue, 1144
OESystem::OEMakeGaussians, 1225
OESystem::OEMakeGridFromCenterAndExtents, 1225
OESystem::OEMakeHex, 1225
OESystem::OEMakeMolecularGaussianGrid, 1226
OESystem::OEMakeRegularGrid, 1226
OESystem::OEMakeRegularSubGrid, 1226
OESystem::OEMakeString, 1227

OESystem::OEMaskGrid, 1227
OESystem::OEMaskGridByCoordsAndRadii, 1227
OESystem::OEMaskGridByMolecule, 1227
OESystem::OEMaskGridValues, 1227
OESystem::OEMemPoolMode, 1190
OESystem::OEMemPoolMode::BoundedCache, 1191
OESystem::OEMemPoolMode::Default, 1191
OESystem::OEMemPoolMode::Mutexed, 1191
OESystem::OEMemPoolMode::SingleThreaded, 1190
OESystem::OEMemPoolMode::Spinlocked, 1191
OESystem::OEMemPoolMode::System, 1191
OESystem::OEMemPoolMode::ThreadLocal, 1191
OESystem::OEMemPoolMode::UnboundedCache, 1191
OESystem::OEMoveArray, 1228
OESystem::OEMTZMapType, 1191
OESystem::OEMTZMapType::Fc, 1191
OESystem::OEMTZMapType::Fdelwt, 1191
OESystem::OEMTZMapType::FiveFoThreeFc, 1191
OESystem::OEMTZMapType::Fo, 1191
OESystem::OEMTZMapType::FoFc, 1192
OESystem::OEMTZMapType::FoFcSigmaA, 1192
OESystem::OEMTZMapType::FoFom, 1192
OESystem::OEMTZMapType::FoSquared, 1192
OESystem::OEMTZMapType::Fwt, 1192
OESystem::OEMTZMapType::None, 1192
OESystem::OEMTZMapType::ThreeFoTwoFc, 1192
OESystem::OEMTZMapType::TwoFoFc, 1192
OESystem::OEMTZMapType::TwoFoFcSigmaA, 1192
OESystem::OEMTZMapType::User, 1192
OESystem::OEMTZMapType::UserDiff, 1192
OESystem::OEMTZMapTypeFromString, 1224
OESystem::OEMTZMapTypeToString, 1224
OESystem::OEMultiGrid, 1144
    Clear, 1146
    Constructors, 1145
    CreateCopy, 1146
    GetDataType, 1146
    IsDataType, 1146
    SetDim, 1146
    SetMid, 1146
    SetSpacing, 1146
    SetValues, 1146
    SetXDim, 1146
    SetXMid, 1147
    SetYDim, 1147
    SetYMid, 1147
    SetZDim, 1147
    SetZMid, 1147
OESystem::OEMultiParameter, 1250
    AddLegalEntry, 1251
    Constructors, 1251
    GetDefault, 1251
    GetSetting, 1251
    GetValue, 1251
    SetDefault, 1251
    SetValue, 1251
OESystem::OEMultiplicationWillOverflow, 1228
OESystem::OEMultiplyGrid, 1228
OESystem::OENot, 1147
    Constructors, 1148
    CreateCopy, 1148
    operator(), 1148
OESystem::OENumberToString, 1229
OESystem::OENumContours, 1228
OESystem::OEOptions, 1252
    AddDetail, 1253
    AddOption, 1253
    AddParameter, 1253
    AddParameterPrefix, 1253
    AddParameterSuffix, 1253
    Clear, 1253
    DeleteOptions, 1253
    DeleteParameter, 1253
    FindDupParameters, 1254
    GetBrief, 1254
    GetDetail, 1254
    GetName, 1254
    GetOption, 1254
    GetOptions, 1254
    GetOrderPriority, 1254
    GetParameter, 1255
    GetParameters, 1255
    HasDupParameters, 1255
    HasOption, 1255
    HasParameter, 1255
    SetBrief, 1255
    SetName, 1255
    SetOrderPriority, 1256
    SetParameterVisibility, 1256
    UpdateValues, 1256
OESystem::OEOptsConfigureStatus, 1261
OESystem::OEOptsConfigureStatus::Failed, 1261
OESystem::OEOptsConfigureStatus::Help, 1261
OESystem::OEOptsConfigureStatus::Success, 1261
OESystem::OEOr, 1148
    Constructors, 1149
    CreateCopy, 1149
    operator(), 1149
OESystem::OEOwnedPtr, 1149
    Constructors, 1150
    Get, 1151
    Release, 1151
OESystem::OEParameter, 1151
    AddAlias, 1152
    AddDetail, 1152
    AddIllegalRange, 1152
    AddIllegalValue, 1152

AddLegalRange, 1152
AddLegalValue, 1153
AddStringDefault, 1153
AddStringValue, 1153
ClearDefaults, 1153
ClearValues, 1153
CreateCopy, 1153
GetAliases, 1153
GetBrief, 1153
GetDetail, 1154
GetHasDefault, 1154
GetHasValue, 1154
GetIllegalRanges, 1154
GetIllegalValues, 1154
GetIsList, 1154
GetKeyless, 1154
GetLegalRanges, 1154
GetLegalValues, 1155
GetName, 1155
GetOrderPriority, 1155
GetPrettyLegalValues, 1155
GetRequired, 1155
GetStringDefault, 1155
GetStringDefaults, 1155
GetStringValue, 1155
GetStringValues, 1156
GetVisibility, 1156
IsLegalString, 1156
IsSet, 1156
IsSetToString, 1156
Load, 1156
LoadDefault, 1156
LoadValue, 1156
SetBrief, 1157
SetIsList, 1157
SetKeyless, 1157
SetName, 1157
SetOrderPriority, 1157
SetRequired, 1157
SetStringDefault, 1157
SetStringValue, 1157
SetVisibility, 1158
OESystem::OEParamVisibility, 1199
OESystem::OEParamVisibility::Hidden, 1200
OESystem::OEParamVisibility::MAX, 1200
OESystem::OEParamVisibility::Normal, 1200
OESystem::OEParamVisibility::Simple, 1200
OESystem::OEParseCommandLine, 1229
OESystem::OEParseCommandLineLW, 1232
OESystem::OEParseHex, 1232
OESystem::OEParseParameterFile, 1233
OESystem::OEPODParameter, 1256
GetDefaultsVector, 1257
GetSettingsVector, 1257
GetValuesVector, 1257
OESystem::OEPointerParameter, 1258
OESystem::OEPredIter, 1158
Constructors, 1158
Copy, 1158
operator bool, 1158
Predicate, 1158
ToFirst, 1159
ToLast, 1159
OESystem::oeprogressstream, 1186
Constructors, 1186
SetTracer, 1186
OESystem::OEProtectedBuffer, 1159
Constructors, 1159
Get, 1159
Null, 1160
Peek, 1160
Put, 1160
OESystem::OERandom, 1160
Constructors, 1160
GetSeed, 1160
NextFloat, 1161
NextInt, 1161
Seed, 1161
TimeSeed, 1161
OESystem::OEReadCIFGridFile, 1233
OESystem::OEReadGrid, 1233
OESystem::OEReadMTZ, 1234
OESystem::OEReadOldStyleGrid, 1234
OESystem::OERegisterMolParameters, 1234
OESystem::OERegisterOEBInitializer, 1235
OESystem::OERole, 1161
Constructors, 1161
GetName, 1162
OESystem::OERoles, 1163
Constructors, 1163
CreateCopy, 1163
OESystem::OERoleSet, 1162
AddRole, 1162
ClearRoles, 1162
DeleteRole, 1162
GetRoles, 1163
HasRole, 1163
NumRoles, 1163
OESystem::OERotate, 1235
OESystem::OEScalarGrid, 1164
Clear, 1165
Constructors, 1165
CreateCopy, 1165
GetDataType, 1166
GetType, 1166
IsDataType, 1166
SetType, 1166
OESystem::OESetMemPoolMode, 1235

OESystem::OESetThreadSafe, 1235
OESystem::OESetupGrid, 1236
OESystem::OESharedPtr, 1166
    Constructors, 1167
    Get, 1167
OESystem::OESigmaNormalizeGrid, 1236
OESystem::OESkewGrid, 1168
    AsRegularGrid, 1168
    Clear, 1169
    Constructors, 1168
    CreateCopy, 1169
    ElementToGridIdx, 1169
    ElementToSpatialCoord, 1169
    GetDataType, 1169
    GetDim, 1169
    GetMid, 1169
    GetReentrant, 1169
    GetSize, 1170
    GetSpaceGroup, 1170
    GetSpacing, 1170
    GetTitle, 1170
    GetType, 1170
    GetUnitCell, 1170
    GetValues, 1170
    GetXDim, 1170
    GetXMid, 1170
    GetYDim, 1171
    GetYMid, 1171
    GetZDim, 1171
    GetZMid, 1171
    HasSpaceGroup, 1171
    HasUnitCell, 1171
    IsDataType, 1171
    IsRegular, 1171
    IsSpacingSet, 1171
    IsTitleSet, 1172
    operator bool, 1168
    SetDim, 1172
    SetMid, 1172
    SetReentrant, 1172
    SetSpaceGroup, 1172
    SetSpacing, 1172
    SetTitle, 1172
    SetType, 1172
    SetUnitCell, 1172
    SetValues, 1173
OESystem::OESNPrintf, 1235
OESystem::OEStaticGetTag, 1236
OESystem::OEStopwatch, 1173
    Constructors, 1173
    Elapsed, 1173
    Lap, 1173
    Start, 1173
OESystem::OEStringCollapseQuotes, 1237
OESystem::OEStringCSVJoin, 1236
OESystem::OEStringCSVQuote, 1236
OESystem::OEStringCSVTokenize, 1237
OESystem::OEStringEquals, 1238
OESystem::OEStringJoin, 1238
OESystem::OEStringLower, 1238
OESystem::OEStringMatch, 1239
OESystem::OEStringParameter, 1258
    Constructors, 1259
OESystem::OEStringSimplifyWhiteSpace, 1239
OESystem::OEStringStripQuotes, 1239
OESystem::OEStringStripWhiteSpace, 1239
OESystem::OEStringTabsToSpaces, 1240
OESystem::OEStringTokenize, 1240
OESystem::OEStringTokenizeQuoted, 1241
OESystem::OEStringToNumber, 1240
OESystem::OEStringTranslate, 1242
OESystem::OEStringUpper, 1242
OESystem::OESubtractGrid, 1242
OESystem::OESystem, 1193
OESystem::OESystem::OEBlack, 1193
OESystem::OESystem::OEBlue, 1193
OESystem::OESystem::OEBlueTint, 1193
OESystem::OESystem::OEBrass, 1193
OESystem::OESystem::OEBrown, 1193
OESystem::OESystem::OECopper, 1194
OESystem::OESystem::OECyan, 1194
OESystem::OESystem::OEDarkBlue, 1194
OESystem::OESystem::OEDarkBrown, 1194
OESystem::OESystem::OEDarkCyan, 1194
OESystem::OESystem::OEDarkGreen, 1194
OESystem::OESystem::OEDarkGrey, 1194
OESystem::OESystem::OEDarkMagenta, 1194
OESystem::OESystem::OEDarkOrange, 1194
OESystem::OESystem::OEDarkPurple, 1194
OESystem::OESystem::OEDarkRed, 1195
OESystem::OESystem::OEDarkRose, 1195
OESystem::OESystem::OEDarkSalmon, 1195
OESystem::OESystem::OEDarkYellow, 1195
OESystem::OESystem::OEGold, 1195
OESystem::OESystem::OEGreen, 1195
OESystem::OESystem::OEGreenBlue, 1195
OESystem::OESystem::OEGreenTint, 1195
OESystem::OESystem::OEGrey, 1195
OESystem::OESystem::OEHotPink, 1195
OESystem::OESystem::OELightBlue, 1196
OESystem::OESystem::OELightBrown, 1196
OESystem::OESystem::OELightGreen, 1196
OESystem::OESystem::OELightGrey, 1196
OESystem::OESystem::OELightOrange, 1196
OESystem::OESystem::OELightPurple, 1196
OESystem::OESystem::OELightSalmon, 1196
OESystem::OESystem::OELimeGreen, 1196
OESystem::OESystem::OEMagenta, 1196

OESystem::OESystem::OEMandarin, 1196
OESystem::OESystem::OEMediumBlue, 1197
OESystem::OESystem::OEMediumBrown, 1197
OESystem::OESystem::OEMediumGreen, 1197
OESystem::OESystem::OEMediumOrange, 1197
OESystem::OESystem::OEMediumPurple, 1197
OESystem::OESystem::OEMediumSalmon, 1197
OESystem::OESystem::OEMediumYellow, 1197
OESystem::OESystem::OEOliveBrown, 1197
OESystem::OESystem::OEOliveGreen, 1197
OESystem::OESystem::OEOliveGrey, 1197
OESystem::OESystem::OEOrange, 1198
OESystem::OESystem::OEPewter, 1198
OESystem::OESystem::OEPink, 1198
OESystem::OESystem::OEPinkTint, 1198
OESystem::OESystem::OEPurple, 1198
OESystem::OESystem::OERed, 1198
OESystem::OESystem::OERedOrange, 1198
OESystem::OESystem::OERoyalBlue, 1198
OESystem::OESystem::OESeaGreen, 1198
OESystem::OESystem::OESilver, 1198
OESystem::OESystem::OESkyBlue, 1199
OESystem::OESystem::OETransparentColor, 1199
OESystem::OESystem::OEViolet, 1199
OESystem::OESystem::OEWhite, 1199
OESystem::OESystem::OEYellow, 1199
OESystem::OESystem::OEYellowTint, 1199
OESystem::OETanimoto, 1242
OESystem::OEThreadedDots, 1173
    Constructors, 1174
    Total, 1174
    Update, 1174
OESystem::OEThrow, 1199
OESystem::OETracerBase, 1175
    Aborted, 1175
    Pop, 1175
    Push, 1175
    Refresh, 1175
    SetProgress, 1175
    SetTask, 1175
OESystem::OETranslate, 1242
OESystem::OETrimToSigFigs, 1242
OESystem::OETypedParameter, 1176
    AddDefault, 1177
    AddValue, 1177
    GetDefault, 1177
    GetDefaults, 1177
    GetSetting, 1177
    GetSettings, 1177
    GetValue, 1177
    GetValues, 1177
    SetDefault, 1177
    SetValue, 1178
OESystem::OEUIntParameter, 1260
    Constructors, 1261
OESystem::OEUnaryFalse, 1178
OESystem::OEUnaryFunction, 1178
    CreateCopy, 1180
    operator(), 1180
OESystem::OEUnaryPredicate, 1180
    CreateCopy, 1183
    CreatePredicateCopy, 1183
    operator(), 1183
OESystem::OEUnaryTrue, 1183
OESystem::OEUnboundedBuffer, 1184
    Constructors, 1184
    Empty, 1184
    Full, 1184
    Get, 1184
    Peek, 1184
    Put, 1184
    Size, 1184
OESystem::OEUnknownOEBDataTagName, 1243
OESystem::OEUnownedPtr, 1185
    Constructors, 1185
    Get, 1185
    Release, 1185
OESystem::OEUpdateOpts, 1263
OESystem::OEWallTimer, 1185
    Constructors, 1185
    Elapsed, 1186
    Lap, 1186
    Start, 1186
OESystem::OEWriteGrid, 1243
OESystem::OEWriteHtml, 1243
OESystem::OEWriteInterfaceConfigurationFile, 1243
OESystem::OEWriteOldStyleGrid, 1243
OESystem::OEWriteSettings, 1243
OEUniMolecularRxn, 117, 120
Open
    OEChem::OEMolDatabase, 567
    OEChem::OESubSearchDatabase, 633
open
    OEChem::oemolstreambase, 589
    OEPlatform::oeistream, 1280
    OEPlatform::oeostream, 1288
OpenByteArray
    OEChem::oemolistream, 574
openfd
    OEPlatform::oeifstream, 1274
    OEPlatform::oeofstream, 1284
openstring
    OEChem::oemolithread, 577
    OEChem::oemolostream, 579
    OEChem::oemolothread, 580
operator bool
    OEBio::OEAltGroup, 1325

OEBio::OEAltLocation, 1326
OEBio::OEAltLocationState, 1333
OEBio::OEDesignUnit, 1346
OEBio::OEInteractionHintContainer, 1369
OEBio::OEIridiumData, 1377
OEBio::OESequenceAlignment, 1464
OEBio::OEStructureQuality, 1473
OEChem::OECliqueSearch, 445
OEChem::OECSVHeader, 461
OEChem::OEFuzzy, 469
OEChem::OELibraryGen, 520
OEChem::OEMatchBase, 531
OEChem::OEMatchFunc, 534
OEChem::OEMCSSearch, 546
OEChem::OEMolBase, 553
OEChem::oemolistream, 571
OEChem::oemolostream, 578
OEChem::oemolthreadbase, 592
OEChem::OEResidue, 615
OEChem::OESubSearch, 626
OEChem::OETrans, 642
OEChem::OEUniMolecularRxn, 650
OEPlatform::OEDirectoryScan, 1265
OEPlatform::oestream, 1290
OESystem::OEConcatIter, 1093
OESystem::OEIter, 1140
OESystem::OEIterBase, 1141
OESystem::OEPredIter, 1158
OESystem::OESkewGrid, 1168
operator()
OEBio::OEAtomMatchResidue, 1334
OEBio::OEHasAlternateLocation, 1350
OEBio::OEHasInteractionHint, 1351
OEBio::OEHasInteractionHintType, 1352
OEBio::OEHasPDBAtomIndex, 1352
OEBio::OEHasResidueInteractionHint, 1353
OEBio::OEHasSecondaryStructure, 1354
OEBio::OEIsBackboneAtom, 1381
OEBio::OEIsCationPiInteractionHint, 1382
OEBio::OEIsChelatorInteractionHint, 1383
OEBio::OEIsClashingHBondInteractionHint, 1384
OEBio::OEIsClashInteractionHint, 1384
OEBio::OEIsContactInteractionHint, 1385
OEBio::OEIsCovalentInteractionHint, 1386
OEBio::OEIsCTerminalAtom, 1387
OEBio::OEIsDisulfideBond, 1387
OEBio::OEIsHalogenBondInteractionHint, 1388
OEBio::OEIsHBondInteractionHint, 1389
OEBio::OEIsHetAtom, 1390
OEBio::OEIsInterInteractionHint, 1391
OEBio::OEIsIntermolecularCationPiInteractionHint, 1391
OEBio::OEIsIntermolecularChelatorInteractionHint, 1392
OEBio::OEIsIntermolecularHalogenBondInteractionHint, 1393
OEBio::OEIsIntermolecularHBondInteractionHint, 1394
OEBio::OEIsIntermolecularSaltBridgeInteractionHint, 1395
OEBio::OEIsIntermolecularStackingInteractionHint, 1396
OEBio::OEIsIntraInteractionHint, 1396
OEBio::OEIsIntramolecularCationPiInteractionHint, 1397
OEBio::OEIsIntramolecularChelatorInteractionHint, 1398
OEBio::OEIsIntramolecularHalogenBondInteractionHint, 1399
OEBio::OEIsIntramolecularHBondInteractionHint, 1400
OEBio::OEIsIntramolecularSaltBridgeInteractionHint, 1401
OEBio::OEIsIntramolecularStackingInteractionHint, 1402
OEBio::OEIsNonIdealHBondInteractionHint, 1402
OEBio::OEIsNTerminalAtom, 1403
OEBio::OEIsNucleicAcidBase, 1404
OEBio::OEIsNucleicAcidPhosphate, 1405
OEBio::OEIsNucleicAcidSugar, 1406
OEBio::OEIsSaltBridgeInteractionHint, 1407
OEBio::OEIsStackingInteractionHint, 1407
OEBio::OEIsStandardAminoAcid, 1408
OEBio::OEIsStandardNucleicAcid, 1409
OEBio::OEIsUnpairedInteractionHint, 1410
OEBio::OEIsUnpairedLigandInteractionHint, 1411
OEBio::OEIsUnpairedProteinInteractionHint, 1411
OEBio::OEIsWater, 1412
OEBio::OEMolComplexCategorizer, 1414
OEBio::OEMolComplexCategorizerBase, 1418
OEBio::OEMolComplexFilter, 1420
OEChem::OEAtomIsInAgent, 418
OEChem::OEAtomIsInChain, 419
OEChem::OEAtomIsInProduct, 419
OEChem::OEAtomIsInReactant, 420
OEChem::OEAtomIsInResidue, 421
OEChem::OEAtomIsInRing, 421
OEChem::OEBondIsInChain, 430
OEChem::OEBondIsInRing, 430
OEChem::OEGetNbrAtom, 470
OEChem::OEHasAlphaBetaUnsat, 475
OEChem::OEHasAminoAcidType, 1019

OEChem::OEHasAnisou, 475
OEChem::OEHasAtomicNum, 476
OEChem::OEHasAtomInGroup, 477
OEChem::OEHasAtomName, 477
OEChem::OEHasAtomStereoSpecified, 478
OEChem::OEHasBondInGroup, 479
OEChem::OEHasBondStereoSpecified, 480
OEChem::OEHasChainID, 480
OEChem::OEHasFormalCharge, 481
OEChem::OEHasFragmentNumber, 482
OEChem::OEHasGroupType, 483
OEChem::OEHasHvyDegree, 483
OEChem::OEHasIdx, 484
OEChem::OEHasMapIdx, 485
OEChem::OEHasMonomerType, 1020
OEChem::OEHasOrder, 486
OEChem::OEHasPolymerType, 1021
OEChem::OEHasResidueNumber, 487
OEChem::OEIdxSelected, 488
OEChem::OEInSamePart, 492
OEChem::OEIsAmideBond, 493
OEChem::OEIsAromaticAtom, 494
OEChem::OEIsAromaticBond, 494
OEChem::OEIsAtomHybridization, 495
OEChem::OEIsBridgeHead, 495
OEChem::OEIsCAlpha, 496
OEChem::OEIsCarbon, 497
OEChem::OEIsChiralAtom, 497
OEChem::OEIsChiralBond, 498
OEChem::OEIsDisulfideBridgeBond, 499
OEChem::OEIsHalide, 499
OEChem::OEIsHalogen, 500
OEChem::OEIsHeavy, 500
OEChem::OEIsHetero, 501
OEChem::OEIsHydrogen, 502
OEChem::OEIsInMonomerCodeSet, 1025
OEChem::OEIsIntraResidueBond, 502
OEChem::OEIsInvertibleNitrogen, 503
OEChem::OEIsMDLStereoGroup, 504
OEChem::OEIsMember, 505
OEChem::OEIsMemberPtr, 506
OEChem::OEIsMetal, 507
OEChem::OEIsNitrogen, 508
OEChem::OEIsNonRingAtomDoubleBondedToRing, 508
OEChem::OEIsOxygen, 510
OEChem::OEIsPhosphorus, 511
OEChem::OEIsPolar, 511
OEChem::OEIsPolarHydrogen, 512
OEChem::OEIsProductGroup, 513
OEChem::OEIsReactantGroup, 513
OEChem::OEIsRGroup, 514
OEChem::OEIsRotor, 515
OEChem::OEIsRxnComponentGroup, 516
OEChem::OEIsSulfur, 517
OEChem::OEIsTermHeavyAtom, 517
OEChem::OEIsValidAtomValence, 518
OEChem::OEIsValidMDLAtomValence, 519
OEChem::OELibraryGen, 520
OEChem::OEMatchFunc, 533
OEChem::OEMCSFunc, 540
OEChem::OEMCSMaxAtoms, 542
OEChem::OEMCSMaxAtomsCompleteCycles, 543
OEChem::OEMCSMaxBonds, 544
OEChem::OEMCSMaxBondsCompleteCycles, 545
OEChem::OENthAtom, 596
OEChem::OEPartPredT, 601
OEChem::OESimilarByRMSD, 622
OEChem::OEUnaryToBinaryAnd, 645
OEChem::OEUnaryToBinaryOr, 646
OEChem::OEUniMolecularRxn, 650
OESystem::OEAnd, 1059
OESystem::OEBinaryAnd, 1070
OESystem::OEBinaryFalse, 1071
OESystem::OEBinaryFunction, 1072
OESystem::OEBinaryNot, 1075
OESystem::OEBinaryOr, 1076
OESystem::OEBinaryPredicate, 1077
OESystem::OEBinaryTrue, 1077
OESystem::OEFixedGrid, 1107
OESystem::OEHasRole, 1130
OESystem::OEIsFalse, 1138
OESystem::OEIsTrue, 1139
OESystem::OENot, 1148
OESystem::OEOr, 1149
OESystem::OEUnaryFunction, 1180
OESystem::OEUnaryPredicate, 1183
operator+=
OESystem::OEBase, 1062
operator=
OEBio::OEDesignUnit, 1345
operator->
OESystem::OEIter, 1139
OESystem::OEIterBase, 1141
Order
OEChem::OEMolDatabase, 567
OrderAtoms
OEChem::OEMolBase, 560
OrderBonds
OEChem::OEAtomBase, 408
OEChem::OEMolBase, 560
OrderConfs
OEChem::OEMCMolBase, 538

P

PAM250, 216
Parent
OESystem::OEInterface, 1137

ParseTitle
    OEChem::OETokenizerBase, 640
partial charge, 44, 79
parts2mols.cs
    Example Code, 246
pattern matching, 85
PDB data, 28
PDB Splitting, 222
Peek
    OESystem::OEBoundedBuffer, 1083
    OESystem::OEProtectedBuffer, 1160
    OESystem::OEUnboundedBuffer, 1184
peekbyte
    OEChem::oemolistream, 575
    OEPlatform::oeistream, 1280
PeekMol
    OEChem::oemolthreadbase, 593
Pop
    OESystem::OEConsoleProgressTracer, 1094
    OESystem::OETracerBase, 1175
PopActive
    OEChem::OEMCMolBase, 538
pread
    OEPlatform::oeifstream, 1275
    OEPlatform::oeiofstream, 1276
Predicate, 217
    OESystem::OEPredIter, 1158
predicate, 38
Predict
    OEMath::OEKernelPLS, 1304
PrevBit
    OESystem::OEBitVector, 1080
printatomnames.cs
    Example Code, 332
printinteractions.cs
    Example Code, 373
proteinprep.cs
    Example Code, 365
Push
    OESystem::OEConcatIter, 1093
    OESystem::OEConsoleProgressTracer, 1094
    OESystem::OEIter, 1140
    OESystem::OEIterBase, 1141
    OESystem::OETracerBase, 1175
PushActive
    OEChem::OEMCMolBase, 539
PushBack
    OEChem::OETrans, 642
PushFront
    OEChem::OETrans, 642
Put
    OESystem::OEBoundedBuffer, 1083
    OESystem::OEProtectedBuffer, 1160
    OESystem::OEUnboundedBuffer, 1184
put
    OEChem::oemolostream, 579
putbyte
    OEPlatform::oeostream, 1288
PutMol
    OEChem::oemolthreadbase, 593

Q

QMol
    OEChem::OEQMol, 609

R

randomizeatoms.cs
    Example Code, 254
randomsample.cs
    Example Code, 270
reaction map index, 44
reaction molecule, 50
reaction role, 44
read
    OEChem::oemolistream, 575
    OEPlatform::oeistream, 1281
Refresh
    OEPlatform::OEFile, 1266
    OESystem::OEConsoleProgressTracer, 1094
    OESystem::OETracerBase, 1175
Release
    OEPlatform::OECondition, 1264
    OEPlatform::OEMutex, 1269
    OEPlatform::OESpinlock, 1271
    OEPlatform::OETryMutex, 1273
    OESystem::OEOwnedPtr, 1151
    OESystem::OEUnownedPtr, 1185
Remove
    OEChem::OEAtomBondSet, 416
RemoveAtom
    OEChem::OEAtomBondSet, 417
    OEChem::OEConfBase, 452
RemoveAtoms
    OEChem::OEAtomBondSet, 417
RemoveBond
    OEChem::OEAtomBase, 408
    OEChem::OEAtomBondSet, 417
    OEChem::OEConfBase, 452
RemoveBonds
    OEChem::OEAtomBondSet, 417
RemoveFromDB
    OEBio::OEResidueCategoryData, 1460
ReplaceReadFxn
    OESystem::OEBinaryIOHandler, 1073
    OESystem::OEBinaryIOHandlerBase, 1074
ReplaceWriteFxn
    OESystem::OEBinaryIOHandler, 1073
    OESystem::OEBinaryIOHandlerBase, 1075

reset
OEPlatform::oeisstream, 1277
ResetFilters
OEBio::OESplitMolComplexOptions, 1468
ResetPerceived
OEChem::OEMolBase, 560
ResetReadFailureCt
OEChem::oemolistream, 572
reshist.cs
Example Code, 351
Residue, 211, 214
Resize
OEMath::OEMatrix, 1306
OEMath::OESquareMatrix, 1308
rewind
OEChem::oemolistream, 575
OEPlatform::oestream, 1290
ring membership, 54
ring processing, 54
ring system identification, 56
ringsubset.cs
Example Code, 264
rmsd.cs
Example Code, 299
Rotamer, 227
rotatable bond, 46
rotorcount.cs
Example Code, 324
Run
OEPlatform::OEThread, 1272

S

Save
OEChem::OEMolDatabase, 567
SCMol
OEChem::OEGraphMol, 471
OEChem::OEMol, 552
OEChem::OEQMol, 609
SD data, 24
sdf2csv.cs
Example Code, 307
sdfilter.cs
Example Code, 314
sdfmodprops.cs
Example Code, 303
sdfrename.cs
Example Code, 312
Search
OEChem::OESubSearchDatabase, 633
Secondary Structure, 216
Seed
OESystem::OERandom, 1161
seek
OEChem::oemolistream, 575
OEPlatform::oestream, 1290
SelectPart
OEChem::OEPartPredT, 601
Sequence Alignment, 216
Set
OEChem::OEIsMember, 505
OEChem::OEIsMemberPtr, 507
OEPlatform::OEFile, 1266
OESystem::OEColor, 1087
set
OEPlatform::oeisstream, 1277
OEPlatform::oeistdstream, 1278
OEPlatform::oeiwrapperstream, 1282
OEPlatform::oeosstream, 1286
OEPlatform::oeostdstream, 1287
OEPlatform::oeowrapperstream, 1289
SetA
OEBio::OECrystalSymmetryParams, 1344
OESystem::OEColor, 1088
SetActive
OEChem::OEMCMolBase, 539
SetAddCarbonSkeleton
OEChem::OE2DRingDictionaryCreatorOptions, 384
SetAddCarbonSkeletonWithBondStereo
OEChem::OE2DRingDictionaryCreatorOptions, 384
SetAddMetadata
OEBio::OEPlaceHydrogensOptions, 1445
SetAddRGroup
OEBio::OESplitMolComplexOptions, 1469
SetAdjustHCount
OEBio::OESplitMolComplexOptions, 1469
SetAll
OESystem::OEGrid, 1119
SetAllowUnmatchedFragments
OEChem::OEHelmGenerationOptions, 1022
SetAllowUnspecifiedStereo
OEChem::OEHelmGenerationOptions, 1022
SetAlpha
OEBio::OECrystalSymmetryParams, 1344
SetAlt
OEBio::OEAltLocationFactory, 1332
SetAltConfs
OEBio::OEIridiumData, 1379
SetAlteredTag
OEChem::OEUniMolecularRxnOptions, 655
SetAlternateLocation
OEBio::OEAtomMatchResidueID, 1337
OEChem::OEResidue, 618
SetAltsMustBeCompatible
OEBio::OEPlaceHydrogensOptions, 1445
SetArg
OEPlatform::OEThread, 1272

SetAromatic
OEChem::OEAtomBase, 408
OEChem::OEBondBase, 427
SetAsDensity
OEBio::OEIridiumData, 1379
SetAssignMapIdx
OEChem::OELibraryGen, 524
SetAssumptions
OEBio::OESplitMolComplexOptions, 1469
SetAtom
OEBio::OEReceptorProteinConstraint, 1459
SetAtomColorer
OEChem::OE3DMolStyle, 392
SetAtomicNum
OEChem::OEAtomBase, 409
SetAtomLabelColorer
OEChem::OE3DMolStyle, 392
SetAtomLabelDataTag
OEChem::OE3DMolStyle, 392
SetAtomLabelText
OEChem::OE3DMolStyle, 392
SetAtomLabelType
OEChem::OE3DMolStyle, 392
SetAtomStyle
OEChem::OE3DMolStyle, 392
SetAxial
OEChem::OEAtomBase, 409
SetB
OEBio::OECrystalSymmetryParams, 1344
OESystem::OEColor, 1088
SetBadClashOverlapDistance
OEBio::OEPlaceHydrogensOptions, 1446
SetBaseData
OESystem::OEBase, 1065
SetBaseWriteFxn
OESystem::OEBinaryIOHandler, 1073
OESystem::OEBinaryIOHandlerBase, 1075
SetBeta
OEBio::OECrystalSymmetryParams, 1344
SetBFactor
OEChem::OEResidue, 618
SetBgn
OEChem::OEBondBase, 427
SetBigStep
OESystem::OEDots, 1097
SetBitOff
OESystem::OEBitVector, 1080
SetBitOn
OESystem::OEBitVector, 1080
SetBoolData
OESystem::OEBase, 1066
SetBrief
OESystem::OEInterface, 1137
OESystem::OEOptions, 1255
OESystem::OEParameter, 1157
SetBundleCofactors
OEBio::OEMolComplexCategorizer, 1416
SetBypassPredicate
OEBio::OEPlaceHydrogensOptions, 1446
SetC
OEBio::OECrystalSymmetryParams, 1344
SetCanon
OEChem::OEParseSmilesOptions, 599
SetCategorizer
OEBio::OESplitMolComplexOptions, 1469
SetCategory
OEBio::OEIridiumData, 1379
OEChem::OECIFData, 439
SetCenter
OEMath::OESphere, 1307
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
SetChainID
OEBio::OEAtomMatchResidueID, 1336
OEChem::OEResidue, 618
SetChiral
OEChem::OEAtomBase, 409
OEChem::OEBondBase, 427
OEChem::OEInChIOptions, 490
SetClearCoordinates
OEChem::OELibraryGen, 524
OEChem::OEUniMolecularRxn, 652
OEChem::OEUniMolecularRxnOptions, 655
SetClearMapping
OEChem::OERxnAutomapperOptions, 1041
SetCodes
OEChem::OEMonomerData, 1032
SetCodeSet
OEChem::OEHelmGenerationOptions, 1023
SetCollectProteinByChain
OEBio::OESplitMolComplexOptions, 1470
SetColor
OESystem::OEAnnotation, 1061
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
SetCommandLine
OESystem::OEHeader, 1132
SetComment
OESystem::OEHeader, 1132
SetConfTest
OEChem::oemolistream, 572
SetContourStyle
OEChem::OE3DMolStyle, 393
SetCoords
OEChem::OEConfBase, 452
OEChem::OEMolBase, 560
SetCustomConstraints
OEBio::OEReceptor, 1455

SetCXSMILES
    OEChem::OEParseSmilesOptions, 600
SetData
    OEChem::OECIFCategory, 434
    OEChem::OECIFData, 440
    OESystem::OEBase, 1066
    OESystem::OEBaseDataT, 1069
    OESystem::OEBitVector, 1080
SetDate
    OESystem::OEHeader, 1132
SetDefault
    OEChem::OEPredicateParameter, 603
    OESystem::OEMultiParameter, 1251
    OESystem::OETypedParameter, 1177
SetDefaultColor
    OEChem::OEMolStyleColorer, 591
SetDetails
    OEBio::OEInteractionHint, 1366
SetDim
    OESystem::OEGrid, 1119
    OESystem::OEMultiGrid, 1146
    OESystem::OESkewGrid, 1172
SetDimension
    OEChem::OEMolBase, 561
SetDoubleData
    OESystem::OEBase, 1066
SetDPI
    OEBio::OEIridiumData, 1379
SetEnabled
    OEBio::OEReceptorConstraintFeature, 1457
    OEBio::OEReceptorProteinConstraint, 1460
SetEnd
    OEChem::OEBondBase, 427
SetEnergy
    OEChem::OEMolBase, 561
SetEntityID
    OEChem::OEResidue, 620
SetExcipients
    OEBio::OEIridiumData, 1379
SetExplicitHydrogens
    OEChem::OELibraryGen, 525
SetExponent
    OESystem::OEExponentColorGradient, 1104
SetExpr
    OEChem::OEQBase, 605
SetExtChainID
    OEChem::OEResidue, 618
SetExtendMappedRegions
    OEChem::OERxnAutomapperOptions, 1041
SetFeatureName
    OEBio::OEReceptorConstraintFeature, 1457
SetFilter
    OEChem::OESubSearchQuery, 634
SetFixedHLayer
    OEChem::OEInChIOptions, 491
SetFixValence
    OEChem::OEUniMolecularRxnOptions, 655
SetFlavor, 21
    OEChem::OEMolDatabaseSaveOptions, 569
    OEChem::oemolistream, 573
    OEChem::oemolithread, 576
    OEChem::oemolostream, 578
    OEChem::oemolothread, 580
    OEChem::oemolstreambase, 588
SetFlipBiasScale
    OEBio::OEPlaceHydrogensOptions, 1446
SetFlipPredicate
    OEBio::OEPlaceHydrogensOptions, 1447
SetFloatData
    OESystem::OEBase, 1066
SetFormalCharge
    OEChem::OEAtomBase, 409
SetFormat, 14
    OEChem::OEMolDatabaseSaveOptions, 569
    OEChem::oemolistream, 573
    OEChem::oemolithread, 576
    OEChem::oemolostream, 578
    OEChem::oemolothread, 580
    OEChem::oemolstreambase, 588
SetFragmentNumber
    OEBio::OEAtomMatchResidueID, 1336
    OEChem::OEResidue, 619
SetG
    OESystem::OEColor, 1087
SetGamma
    OEBio::OECrystalSymmetryParams, 1344
SetGridContourColorer
    OEChem::OE3DMolStyle, 393
SetGridType
    OEChem::OE3DMolStyle, 393
SetGroupID
    OEBio::OEAltGroup, 1326
SetGroupType
    OEChem::OEGroupBase, 474
Setgz
    OEChem::oemolistream, 573
    OEChem::oemolostream, 578
SetHandlerImpl
    OESystem::OEErrorHandler, 1100
SetHasMTZ
    OEBio::OEIridiumData, 1379
SetHetAtom
    OEChem::OEResidue, 619
SetHyb
    OEChem::OEAtomBase, 409
SetHydrogenConversions
    OEChem::OEUniMolecularRxnOptions, 656
SetHydrogens

OEChem::OEInChIOptions, 491
SetHydrogenVisibility
OEChem::OE3DMolStyle, 393
SetIdxFileName
OEChem::OEMolDatabaseSaveOptions, 569
SetIgnoreHydrogens
OEChem::OEConfRMSDOptions, 456
SetIgnoreTerminalHeavy
OEChem::OEConfRMSDOptions, 456
SetImplicitHCount
OEChem::OEAtomBase, 410
SetInFile
OEChem::OERefInputAppOptions, 614
OEChem::OESimpleAppOptions, 625
SetInnerContourLevel
OEBio::OEReceptor, 1455
SetInRing
OEChem::OEAtomBase, 410
OEChem::OEBondBase, 428
SetInsertCode
OEBio::OEAtomMatchResidueID, 1336
OEChem::OEResidue, 619
SetIntData
OESystem::OEBase, 1067
SetInteractionStyle
OEChem::OE3DMolStyle, 394
SetInterface
OESystem::OEHeader, 1132
SetIntType
OEChem::OEAtomBase, 410
OEChem::OEBondBase, 428
SetIridiumData
OEBio::OEStructureQuality, 1474
SetIrrationalRfree
OEBio::OEIridiumData, 1380
SetIsList
OESystem::OEParameter, 1157
SetIsotope
OEChem::OEAtomBase, 410
SetKeepTitle
OEChem::OECreateSubSearchDatabaseOptions, 460
SetKeyless
OESystem::OEParameter, 1157
SetLevel
OESystem::OEErrorHandler, 1100
OESystem::OEErrorHandlerImplBase, 1102
SetLigandDensity
OEBio::OEIridiumData, 1380
SetLigandFilter
OEBio::OESplitMolComplexOptions, 1470
SetLigandSubSearch
OEBio::OEMolComplexCategorizer, 1417
SetLocationID
OEBio::OEAltLocation, 1327
SetMapIdx
OEChem::OEAtomBase, 410
SetMappingMethod
OEChem::OERxnAutomapperOptions, 1041
SetMaxAcceptorAngle
OEBio::OEPerceiveInteractionOptions, 1425
SetMaxBindingSiteDist
OEBio::OESplitMolComplexOptions, 1470
SetMaxCationPiAngle
OEBio::OEPerceiveInteractionOptions, 1425
SetMaxCationPiDistance
OEBio::OEPerceiveInteractionOptions, 1426
SetMaxChargeAidedHBondDistance
OEBio::OEPerceiveInteractionOptions, 1426
SetMaxContactFraction
OEBio::OEPerceiveInteractionOptions, 1426
SetMaxCovalentFraction
OEBio::OEPerceiveInteractionOptions, 1427
SetMaxDonorAngle
OEBio::OEPerceiveInteractionOptions, 1427
SetMaxHalogenAngle
OEBio::OEPerceiveInteractionOptions, 1428
SetMaxHalogenBondDistance
OEBio::OEPerceiveInteractionOptions, 1428
SetMaxHBondDistance
OEBio::OEPerceiveInteractionOptions, 1428
SetMaxHeavyAtom
OEBio::OELigandInteractionHintComponent, 1413
SetMaxLigAtoms
OEBio::OEMolComplexCategorizer, 1417
SetMaxLigResidues
OEBio::OEMolComplexCategorizer, 1417
SetMaxMatches
OEChem::OECliqueSearch, 447
OEChem::OEConfRMSDOptions, 456
OEChem::OEMCSSearch, 549
OEChem::OESubSearch, 628
OEChem::OESubSearchQuery, 635
OEChem::OEUniMolecularRxnOptions, 656
SetMaxMetalBondDistance
OEBio::OEPerceiveInteractionOptions, 1429
SetMaxNonIdealAcceptorAngle
OEBio::OEPerceiveInteractionOptions, 1429
SetMaxNonIdealDonorAngle
OEBio::OEPerceiveInteractionOptions, 1429
SetMaxNonIdealHBondDistance
OEBio::OEPerceiveInteractionOptions, 1429
SetMaxPeptideResidues
OEBio::OEMolComplexCategorizer, 1417
SetMaxPiStackAngle
OEBio::OEPerceiveInteractionOptions, 1429
SetMaxPiStackDistance

OEBio::OEPerceiveInteractionOptions, 1430
SetMaxProteinAtoms
OEBio::OEMolComplexCategorizer, 1417
SetMaxProteinIntraDistance
OEBio::OEPerceiveInteractionOptions, 1430
SetMaxSaltBridgeDistance
OEBio::OEPerceiveInteractionOptions, 1430
SetMaxSubstateCutoff
OEBio::OEPlaceHydrogensOptions, 1447
SetMaxSurfaceWaterDist
OEBio::OESplitMolComplexOptions, 1470
SetMaxTStackAngle
OEBio::OEPerceiveInteractionOptions, 1430
SetMaxTStackDistance
OEBio::OEPerceiveInteractionOptions, 1431
SetMaxUnpairedAngle
OEBio::OEPerceiveInteractionOptions, 1431
SetMaxUnpairedDistance
OEBio::OEPerceiveInteractionOptions, 1431
SetMCSExhaustiveMode
OEChem::OERxnAutomapperOptions, 1041
SetMCSFunc
OEChem::OECliqueSearch, 447
OEChem::OEMCSSearch, 549
SetMid
OESystem::OEGrid, 1119
OESystem::OEMultiGrid, 1146
OESystem::OESkewGrid, 1172
SetMinAtoms
OEChem::OECliqueSearch, 447
OEChem::OEMCSSearch, 549
SetMinContactFraction
OEBio::OEPerceiveInteractionOptions, 1432
SetMinCovalentFraction
OEBio::OEPerceiveInteractionOptions, 1432
SetMinHBondContactFraction
OEBio::OEPerceiveInteractionOptions, 1433
SetMinLigAtoms
OEBio::OEMolComplexCategorizer, 1418
SetMinStackOffset
OEBio::OEPerceiveInteractionOptions, 1433
SetMinUnmatchedAtomsRatio
OEChem::OEHelmGenerationOptions, 1023
SetMMCIFChemCompData
OEChem::OECIFData, 440
SetModelNumber
OEChem::OEResidue, 619
SetMonomerType
OEChem::OEMonomerData, 1032
SetName
OEBio::OEAtomMatchResidueID, 1336
OEBio::OEReceptorProteinConstraint, 1459
OEChem::OEAtomBase, 411
OEChem::OEMonomerData, 1032
OEChem::OEResidue, 619
OESystem::OEInterface, 1137
OESystem::OEOptions, 1255
OESystem::OEParameter, 1157
SetNoFlipPredicate
OEBio::OEPlaceHydrogensOptions, 1447
SetNonPolymerPseudoAtoms
OEChem::OEInChIOptions, 491
SetNumGrids
OESystem::OEFixedMultiGrid, 1109
SetNumProcessors
OEChem::OECreateSubSearchDatabaseOptions, 460
SetOccupancy
OEChem::OEResidue, 620
SetOERelease
OESystem::OEHeader, 1132
SetOEVersion
OESystem::OEHeader, 1133
SetOrder
OEChem::OEBondBase, 428
SetOrderPriority
OESystem::OEInterface, 1137
OESystem::OEOptions, 1256
OESystem::OEParameter, 1157
SetOuterContourLevel
OEBio::OEReceptor, 1455
SetOutFile
OEChem::OERefInputAppOptions, 614
OEChem::OESimpleAppOptions, 625
SetOutputStream
OESystem::OEErrorHandler, 1100
SetPackedColor
OESystem::OEColor, 1089
SetPackingResidues
OEBio::OEIridiumData, 1380
SetParameterVisibility
OESystem::OEOptions, 1256
SetPartialCharge
OEChem::OEAtomBase, 411
SetPartOccupancyLigand
OEBio::OEIridiumData, 1380
SetPartOccupancyProtein
OEBio::OEIridiumData, 1380
SetPerceiveCell
OEChem::OECIFOptions, 443
SetPerceiveChemComp
OEChem::OECIFOptions, 443
SetPerceived
OEChem::OEMolBase, 561
SetPerceiveEntity
OEChem::OECIFOptions, 443
SetPerceiveExperimental
OEChem::OECIFOptions, 443

SetPerceiveOther
OEChem::OECIFOptions, 444
SetPerceiveStruct
OEChem::OECIFOptions, 443
SetPerceiveSymmetry
OEChem::OECIFOptions, 443
SetPhi
OEChem::OEEuler, 464
SetPlatform
OESystem::OEHeader, 1133
SetPolymerType
OEChem::OEMonomerData, 1032
SetPossibleCovalent
OEBio::OEIridiumData, 1380
SetPrefactor
OESystem::OEGaussian, 1111
OESystem::OEGaussianBase, 1113
SetPrimaryAlt
OEBio::OEAltLocationFactory, 1332
SetProgram
OESystem::OEHeader, 1133
SetProgress
OESystem::OEConsoleProgressTracer, 1094
OESystem::OETracerBase, 1175
SetPromoteUnknowns
OEBio::OESplitMolComplexOptions, 1470
SetProteinColorer
OEChem::OE3DMolStyle, 394
SetProteinFilter
OEBio::OESplitMolComplexOptions, 1470
SetProteinStyle
OEChem::OE3DMolStyle, 394
SetPsi
OEChem::OEEuler, 464
SetQuiet
OEChem::OEParseSmilesOptions, 600
SetR
OESystem::OEColor, 1087
SetRacemicStereo
OEChem::OEInChIOptions, 491
SetRad
OEMath::OESphere, 1307
SetRadius
OEChem::OEAtomBase, 411
SetRangeOff
OESystem::OEBitVector, 1081
SetRangeOn
OESystem::OEBitVector, 1081
SetReadFxn
OESystem::OEBinaryIOHandler, 1073
OESystem::OEBinaryIOHandlerBase, 1075
SetReconnectedMetals
OEChem::OEInChIOptions, 491
SetReentrant
OESystem::OESkewGrid, 1172
SetRefFile
OEChem::OERefInputAppOptions, 614
SetRelativeStereo
OEChem::OEInChIOptions, 491
SetRemoveUnmappedFragments
OEChem::OELibraryGen, 525
SetRenderHint
OESystem::OEAnnotation, 1061
SetRequired
OESystem::OEParameter, 1157
SetResidueCategoryData
OEBio::OEMolComplexCategorizer, 1418
SetResidueNumber
OEBio::OEAtomMatchResidueID, 1336
OEChem::OEResidue, 620
SetResolution
OEBio::OEIridiumData, 1380
SetRetainBondStereo
OEChem::OEExtractRingTemplateOptions, 466
SetRetainExistingBuiltInTemplates
OEChem::OE2DRingDictionaryCreatorOptions, 385
SetRetainExistingTemplates
OEChem::OE2DRingDictionaryCreatorOptions, 385
OEChem::OEExtractRingTemplateOptions, 467
SetReturn
OEPlatform::OEThread, 1272
SetRFree
OEBio::OEIridiumData, 1380
SetRGBA
OESystem::OEColor, 1089
SetRowData
OEMath::OEMatrix, 1306
SetRxn
OEChem::OEMolBase, 562
SetRxnRole
OEChem::OEAtomBase, 411
SetSaveRange
OEChem::OECliqueSearch, 448
SetSecondaryStructure
OEChem::OEResidue, 620
SetSeparateResidues
OEBio::OESplitMolComplexOptions, 1471
SetSequenceID
OEChem::OEResidue, 620
SetSerialNumber
OEChem::OEResidue, 620
SetSiteResidues
OEBio::OEDesignUnit, 1347
SetSize
OESystem::OEBitVector, 1081
SetSmallStep

OESystem::OEDots, 1097
SetSmiles
OEChem::OEMonomerData, 1033
SetSortByBitCounts
OEChem::OECreateSubSearchDatabaseOptions, 461
SetSpaceGroup
OEBio::OECrystalSymmetryParams, 1345
OESystem::OESkewGrid, 1172
SetSpacing
OESystem::OEGrid, 1120
OESystem::OEMultiGrid, 1146
OESystem::OESkewGrid, 1172
SetSplitCovalent
OEBio::OESplitMolComplexOptions, 1471
SetSplitCovalentCofactors
OEBio::OESplitMolComplexOptions, 1471
SetStackSize
OEPlatform::OEThread, 1272
SetStandardizeBondLen
OEBio::OEPlaceHydrogensOptions, 1448
SetStartingMaterial
OEChem::OELibraryGen, 525
SetState
OEBio::OEAltLocationFactory, 1332
SetStereo
OEChem::OEAtomBase, 411
OEChem::OEBondBase, 428
OEChem::OEInChIOptions, 491
SetStrict
OEChem::OEParseSmilesOptions, 600
OESystem::OEErrorHandler, 1100
OESystem::OEErrorHandlerImplBase, 1102
SetStrictMapping
OEChem::OERxnAutomapperOptions, 1041
SetStrictSmirks
OEChem::OEUniMolecularRxnOptions, 656
SetString
OEChem::oemolostream, 578
SetStringData
OESystem::OEBase, 1067
SetStringDefault
OESystem::OEParameter, 1157
SetStringValue
OESystem::OEParameter, 1157
SetSubChainID
OEChem::OEResidue, 619
SetSurfaceAlpha
OEChem::OE3DMolStyle, 394
SetSurfaceColorer
OEChem::OE3DMolStyle, 394
SetSurfaceStyle
OEChem::OE3DMolStyle, 395
SetSurfaceType
OEChem::OE3DMolStyle, 395
SetSymmetryClass
OEChem::OEAtomBase, 412
SetTag
OEChem::OESDDataPair, 622
SetTask
OESystem::OEConsoleProgressTracer, 1094
OESystem::OETracerBase, 1175
SetText
OESystem::OEColor, 1089
SetTheta
OEChem::OEEuler, 464
SetTimeout
OEChem::OEInChIOptions, 492
SetTitle
OEBio::OEDesignUnit, 1347
OEBio::OEInteractionHintContainer, 1372
OEChem::OEMolBase, 562
OESystem::OEAnnotation, 1061
OESystem::OEGridAbstractBase, 1122
OESystem::OEGridBase, 1129
OESystem::OESkewGrid, 1172
SetTitleSeparator
OEChem::OELibraryGen, 526
SetTorsion
OEChem::OEConfBase, 453
SetTracer
OEChem::oemolistream, 573
OESystem::oeprogressstream, 1186
SetType
OEBio::OEReceptorProteinConstraint, 1459
OEChem::OEAtomBase, 412
OEChem::OEBondBase, 429
OESystem::OEAnnotation, 1061
OESystem::OEScalarGrid, 1166
OESystem::OESkewGrid, 1172
SetU11
OEChem::OEAnisoUij, 398
SetU12
OEChem::OEAnisoUij, 398
SetU13
OEChem::OEAnisoUij, 398
SetU22
OEChem::OEAnisoUij, 399
SetU23
OEChem::OEAnisoUij, 399
SetU33
OEChem::OEAnisoUij, 399
SetUndefinedStereo
OEChem::OEInChIOptions, 492
SetUnitCell
OESystem::OESkewGrid, 1172
Setup
OEMath::OEBox, 1303

SetValenceCorrection
OEChem::OELibraryGen, 526
SetValidateKekule
OEChem::OELibraryGen, 526
OEChem::OEUniMolecularRxn, 652
OEChem::OEUniMolecularRxnOptions, 656
SetValue
OEChem::OEMolParameter, 582
OEChem::OEPredicateParameter, 603
OEChem::OESDDataPair, 622
OESystem::OEMultiParameter, 1251
OESystem::OETypedParameter, 1178
SetValues
OESystem::OEFixedGrid, 1108
OESystem::OEMultiGrid, 1146
OESystem::OESkewGrid, 1173
SetVersion
OEChem::OEMonomerSet, 1037
OESystem::OEHeader, 1133
SetVisibility
OESystem::OEParameter, 1158
SetWarnNoLigand
OEBio::OESplitMolComplexOptions, 1471
SetWaterFilter
OEBio::OESplitMolComplexOptions, 1471
SetWaterProcessing
OEBio::OEPlaceHydrogensOptions, 1448
SetWeight
OEBio::OEInteractionHint, 1366
SetWidth
OESystem::OEGaussian, 1112
OESystem::OEGaussianBase, 1113
SetWriteFxn
OESystem::OEBinaryIOHandler, 1073
OESystem::OEBinaryIOHandlerBase, 1075
SetWriteIdx
OEChem::OEMolDatabaseSaveOptions, 570
SetX
OESystem::OEGaussian, 1112
OESystem::OEGaussianBase, 1113
SetXDim
OESystem::OEGrid, 1120
OESystem::OEMultiGrid, 1146
SetXMid
OESystem::OEGrid, 1120
OESystem::OEMultiGrid, 1147
SetY
OESystem::OEGaussian, 1112
OESystem::OEGaussianBase, 1114
SetYDim
OESystem::OEGrid, 1120
OESystem::OEMultiGrid, 1147
SetYMid
OESystem::OEGrid, 1120
OESystem::OEMultiGrid, 1147
SetZ
OESystem::OEGaussian, 1112
OESystem::OEGaussianBase, 1114
SetZDim
OESystem::OEGrid, 1120
OESystem::OEMultiGrid, 1147
SetZMid
OESystem::OEGrid, 1120
OESystem::OEMultiGrid, 1147
SetZValue
OEBio::OECrystalSymmetryParams, 1345
Sidechain Rotamer, 227
Signal
OEPlatform::OECondition, 1264
Similarity
OEChem::OELingoSim, 528
single atom index, 44
SingleMatch
OEChem::OESubSearch, 629
Size
OEBio::OEResidueCategoryData, 1461
OEPlatform::OEFile, 1266
OESystem::OEBoundedBuffer, 1083
OESystem::OEUnboundedBuffer, 1184
size
OEChem::oemolistream, 575
OEPlatform::oestream, 1291
sizefilter.cs
Example Code, 259
skip
OEChem::oemolistream, 575
OEPlatform::oeistream, 1281
skipbyte
OEChem::oemolistream, 575
OEPlatform::oeistream, 1281
skipline
OEChem::oemolistream, 575
OEPlatform::oeistream, 1281
smallest ring membership, 55
smallest set of smallest rings, 54
SMARTS, 85, 180
SMARTS atom primitives, 181
SMARTS bond primitives, 182
SMARTS logical operations, 182
smartsalign.cs
Example Code, 296
SMILES, 3, 178
Sort
OESystem::OEIter, 1140
SortByAttribute
OEChem::OECIFCategory, 434
Space Group, 216
SpatialCoordToElement

OESystem::OEGridBase, 1129
SpatialCoordToGridIdx
OESystem::OEGridBase, 1129
splitmolcomplex.cs
Example Code, 354
splitmolcomplexfrags.cs
Example Code, 361
splitmolcomplexlowlevel.cs
Example Code, 357
SSSR, 54, 58
Start
OEPlatform::OEThread, 1272
OESystem::OECycleTimer, 1096
OESystem::OEStopwatch, 1173
OESystem::OEWallTimer, 1186
str
OEPlatform::oeosstream, 1286
stream
OEPlatform::oeistdstream, 1279
OEPlatform::oeiwrapperstream, 1282
OEPlatform::oeostdstream, 1287
OEPlatform::oeowrapperstream, 1289
stripsalts.cs
Example Code, 262
substructure search, 85
Swap Ambiguous, 227
SwapEnds
OEChem::OEBondBase, 429
Sweep
OEChem::OEAtomBase, 412
OEChem::OEAtomBondSet, 417
OEChem::OEBondBase, 429
OEChem::OEGroupBase, 474
OEChem::OEMolBase, 562
SweepConfs
OEChem::OEMCMolBase, 539
Symmetry, 216

T

tell
OEChem::oemolistream, 575
OEChem::oemolostream, 579
OEPlatform::oestream, 1291
TimeSeed
OESystem::OERandom, 1161
ToFirst
OESystem::OEConcatIter, 1093
OESystem::OEIter, 1140
OESystem::OEIterBase, 1141
OESystem::OEPredIter, 1159
ToggleBit
OESystem::OEBitVector, 1082
ToHexString
OEBio::OEAltLocationState, 1333
OESystem::OEBitVector, 1081
ToLast
OESystem::OEConcatIter, 1093
OESystem::OEIter, 1140
OESystem::OEIterBase, 1141
OESystem::OEPredIter, 1159
Torsion, 227
Total
OESystem::OEDots, 1097
OESystem::OEThreadedDots, 1174
total hydrogen count, 44
Transform
OEChem::OEConfBase, 453
OEChem::OETrans, 642
OEChem::OETransBase, 643
TransformValue
OESystem::OEColorGradientBase, 1091
OESystem::OEExponentColorGradient, 1104
OESystem::OEExponentialColorGradient, 1106
OESystem::OELinearColorGradient, 1143
OESystem::OELogarithmicColorGradient, 1144
Translate
OEChem::OEAtomBondSet, 417
Transpose
OEMath::OEMatrix, 1306
Try
OEPlatform::OETryMutex, 1273

U

UnCompress
OEChem::OEMolBase, 563
unimolrxn.cs
Example Code, 278
uniqinchi.cs
Example Code, 252
uniqmol.cs
Example Code, 251
Unit Cell, 216
Update
OEChem::OECIFData, 440
OESystem::OEDots, 1097
OESystem::OEThreadedDots, 1174
UpdateValues
OESystem::OEOptions, 1256
Usage
OESystem::OEErrorHandler, 1101
UsingExplicitLigandName
OEBio::OEMolComplexCategorizer, 1418
OEBio::OEMolComplexCategorizerBase, 1419

V

valence models, 47

MDL valence model, 47
OpenEye charge valence model, 49
OpenEye hydrogen count valence model, 50
Validate
OEChem::OEMonomerData, 1033
ValidateMMCIFHeader
OEChem::OECIFData, 441
ValidateMMCIFMol
OEChem::OECIFData, 441
Verbose
OESystem::OEErrorHandler, 1101

W

Wait
OEPlatform::OECondition, 1264
Warning
OESystem::OEErrorHandler, 1101
what
OEChem::OEMolDBException, 570
write
OEChem::oemolostream, 579
OEPlatform::oeostream, 1288
WriteMolecule
OEChem::OEMolDatabase, 568