



OpenEye
Scientific Software

OEDocking TK – C++
Release 1.1.1

OpenEye Scientific Software, Inc.

January 11, 2012

CONTENTS

| | | |
|----------|--|-----------|
| 1 | Front Matter | 1 |
| 2 | Introduction | 3 |
| 2.1 | Welcome to the Docking Toolkit documentation | 3 |
| 2.2 | Getting Started | 3 |
| 3 | Docking Theory | 5 |
| 3.1 | Receptors | 5 |
| 3.2 | Scoring | 13 |
| 3.3 | Docking | 20 |
| 4 | API | 27 |
| 4.1 | OEDocking Classes | 27 |
| 4.2 | OEDocking Constants | 50 |
| 4.3 | OEDocking Functions | 56 |
| 5 | Release Notes | 75 |
| 5.1 | DockingTK 1.1.1 | 75 |
| 5.2 | DockingTK 1.1.0 | 75 |
| 5.3 | DockingTK 1.0.0 | 76 |
| 6 | Bibliography | 77 |
| | Bibliography | 79 |
| | Index | 81 |

FRONT MATTER

Copyright 1997-2012 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. 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 OpenEye Scientific Software.

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 OpenEye Scientific Software's 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 OpenEye.

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. OpenEye Scientific Software, 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, and Macintosh are registered trademarks of Apple Computer, Inc. AIX and IBM are registered trademarks of International Business Machines Corporation. UNIX is a registered trademark of the Open Group. RedHat is a registered trademark of RedHat, Inc. Linux is a registered trademark of Linus Torvalds. SPARC is a registered trademark of SPARC International Inc.

SYBYL is a registered trademark of TRIPOS, Inc. MDL is a registered trademark and ISIS is a trademark of Accelrys, Inc. SMILES, SMARTS, and SMIRKS may be trademarks of Daylight Chemical Information Systems. Macromodel is a trademark of Schrodinger, Inc.

Python is a trademark of the Python Software Foundation. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

INTRODUCTION

2.1 Welcome to the Docking Toolkit documentation

The *Docking Toolkit* library provides a facility for docking and scoring molecules in the context of a protein active site.

The following basic functionalities are available:

- Docking (see *Docking* chapter)
- Scoring (see *Scoring* chapter)

The aim of this manual is to familiarize the user with the *Docking Toolkit* functionalities, however, it does not provide explanations of basic *OEChem* classes and functions. Therefore reading the *OEChem* manual beforehand is highly recommended.

2.2 Getting Started

To include OpenEye libraries in a C++ program the appropriate header files must be included at compile time. All OpenEye libraries require that the `openeye.h` header file is included first. Then additional OpenEye libraries can be included, for example, the *DockingTK* library:

```
#include "openeye.h"  
#include "oedocking.h"
```

Every OpenEye library resides in its own C++ namespace. It is often useful to use the following idiom to produce shorter, more readable code:

```
using namespace OEDocking;
```

Since the *DockingTK*'s objects and functions have unique names, there is little chance to have a name clash with this particular idiom.

To compile the example programs, edit the `CXX =` line in each Makefile. The `CXX` variable must point to the location of the compiler on your system that is equivalent to the compiler used to make the distribution. If the distribution was made with the native compiler for the system then this variable should already be set correctly. Once this has been updated, type `make` in the `openeye/toolkits/examples` directory to build the examples. Several of the example programs require the `text2hex` program that resides in this directory; therefore, this program needs to be built before running `make` in the subdirectories. (Running `make` in the `openeye/toolkits/examples` directory first will take care of this.)

DOCKING THEORY

3.1 Receptors

In the *Docking Toolkit* receptor objects contain the structure of a target protein and information about the location and characteristics of the binding pocket. The receptor can be used in both docking (see *Docking*) and scoring (see *Scoring*).

A receptor is an `OEMolBase` object with the structure of a target protein that has additional information tagged to it describing the characteristics of the active site. The structure of the target protein is accessible via the standard `OEChem::OEMolBase` API. The additional active site information is accessible via an API comprised of a set of free functions (see *Receptor API* section).

A receptor **always** contains:

- The structure of a target protein
- A negative image that describes the shape of the active site

A receptor **may** contain:

- The structure of a ligand bound to the active site
- Docking constraints specifying required protein-ligand interactions
- Extra molecules that do not affect either docking or scoring
 - Generally water or other solvent molecules

3.1.1 Receptor I/O

Within the *Docking Toolkit* receptors are `OEGraphMol` objects (or any other `OEMolBase` object). On disk receptors are stored in OEB format (and thus have `.oeb` or `.oeb.gz` extensions).

The following code snippet reads a receptor

```
OEGraphMol receptor;  
string inputReceptorFilename = argv[1];  
OEReadReceptorFile(receptor, inputReceptorFilename);
```

Note: A receptor can also be read using the `oechem` `OEReadMolecule` function. The difference in behavior is that `OEReadReceptorFile` will fail and return false if the molecule being read is not a receptor.

A receptor can be written to disk as follows

```
string outputReceptorFilename = argv[2];
OEWriteReceptorFile(receptor, outputReceptorFilename);
```

Note: A receptor can also be written using the `oechem OEWriteMolecule` command, however writing to any format besides `OEBinary` will result in **all receptor information being lost except for the protein structure**.

See Also:

Receptor file I/O API documentation

3.1.2 Creating a Receptor

Within the Docking toolkit the `OEMakeReceptor` function is used to create a receptor from the structure of a target protein and one of the following:

Box

```
oemolistream imstr(argv[1]);
OEGraphMol proteinStructure;
OEReadMolecule(imstr, proteinStructure);

OEBox box(xmax, ymax, zmax, xmin, ymin, zmin);
OEGraphMol receptor;
OEMakeReceptor(receptor, proteinStructure, box);
```

The box should enclose the active site. All molecules docked into the receptor or being scored will be required to fit within the box (*i.e.* all heavy atom centers must be within the box).

Bound Ligand

```
OEGraphMol protein;
OEGraphMol ligand;
OEGraphMol receptor;

OEReadMolecule(proteinMolStream, protein);
OEReadMolecule(ligandMolStream, ligand);

OEMakeReceptor(receptor, protein, ligand);
```

The created receptor will include the ligand as a bound ligand.

Hint coordinate

```
OEGraphMol protein;
oemolistream imstr(argv[1]);
OEReadMolecule(imstr, protein);

float hintX, hintY, hintZ;
OEStringToNumber(argv[2], hintX);
OEStringToNumber(argv[3], hintY);
OEStringToNumber(argv[4], hintZ);

OEGraphMol receptor;
OEMakeReceptor(receptor, protein, hintX, hintY, hintZ);
```

The hint coordinate should be in or near the receptor pocket.

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 stripped from the protein structure passed to the `OEMakeReceptor` function, although in certain cases, the user may wish to retain certain key molecules as part of the protein structure (e.g. a crystallographic water).

Note: Receptors can also be created with the *FRED* application or the *Fred Receptor* GUI wizard. Receptor creation using either the *FRED* application or GUI wizard is covered by the *FRED* documentation.

3.1.3 Contents of a Receptor

Protein Structure

The protein structure includes all molecules that a ligand being docked or scored interacts with. This generally only includes the protein, but may also include key cofactors or solvent molecules (e.g. a crystallographic water). All parts of the receptor structure are static during both docking and scoring and any cofactors or solvent molecules in the receptor structure are not displaceable.

Note: A receptor may contain bound ligands and extra molecules. These molecules are part of the receptor object as a whole, but not the protein structure.

A receptor is a `OEMolBase` object with the structure of a target protein. The protein structure is accessible via the API of `OEMolBase`, while all other receptor information is accessed via a specialized set of free functions (see *Receptor API* section). For example, calling the method `OEMolBase::GetAtoms` on a receptor will return atoms of the protein structure, but not the atoms of the bound ligand or extra molecules.

Warning: Modification to the receptor structure will not automatically update all other parts of the receptor. If receptor modifications change the shape of the active site the negative image will no longer correctly describe the shape of the active site. Also, if the coordinate system of the receptor structure is changed, neither the negative image nor any custom constraints will be valid.

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.

The negative image of the receptor is setup when the receptor is created with the `OEMakeReceptor` function (see *Creating a Receptor* section). 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). Thus either the inner or outer contour can be toggled on and off by multiplying the respective level by -1 as shown in the following code snippet.

```
float innerContourLevel = OEReceptorGetInnerContourLevel(receptor);
OEReceptorSetInnerContourLevel(receptor, -innerContourLevel);
```

The above example assumes that the absolute contour level is set to a reasonable value.

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.

The following code example reports the volume of the outer contour.

```
OEScalarGrid negativeImagePotential = OEReceptorGetNegativeImageGrid(receptor);
float outerContourLevel = OEReceptorGetOuterContourLevel(receptor);

unsigned int outerCount = 0;
for (unsigned int i=0 ; i<negativeImagePotential.GetSize() ; ++i)
    if (negativeImagePotential[i] >= outerContourLevel)
        ++outerCount;

float countToVolume = pow(negativeImagePotential.GetSpacing(), 3);

cout << outerCount * countToVolume << " cubic angstroms" << endl;
```

This example sets the outer contour volume to a specified volume in cubic angstroms.

```
OEScalarGrid negativeImagePotential = OEReceptorGetNegativeImageGrid(receptor);

vector<float> gridElement;
for (unsigned int i=0 ; i<negativeImagePotential.GetSize() ; ++i)
    gridElement.push_back(negativeImagePotential[i]);
sort(gridElement.begin(), gridElement.end(), greater<float>());

float outerContourLevel = gridElement[gridElement.size()-1];
float countToVolume = pow(negativeImagePotential.GetSpacing(), 3);
unsigned int ilevel = (unsigned int) (outerContourVolume/countToVolume + 0.5f);
if (ilevel < gridElement.size())
    outerContourLevel = gridElement[ilevel];

OEReceptorSetOuterContourLevel(receptor, outerContourLevel);
```

See Also:

- `OEReceptorGetNegativeImageGrid`
- `OEReceptorHasOuterContourLevel`
- `OEReceptorGetOuterContourLevel`

- `OEReceptorSetOuterContourLevel`
- `OEReceptorHasInnerContourLevel`
- `OEReceptorGetInnerContourLevel`
- `OEReceptorSetInnerContourLevel`

Bound Ligand

A receptor may optionally contain the structure of a single ligand bound into the active site. A receptor is required to have a bound ligand when using the hybrid docking method (see *Hybrid Method* section), but it is ignored by all other docking and scoring methods. Typically the structure is obtained experimentally by X-ray crystallography along with the protein structure, although the ligand structure can be determined by other means (*e.g.* docking).

The following code snippet checks to see if a receptor has a bound ligand

```
if (OEReceptorHasBoundLigand(receptor))
    cout << "Receptor has a bound ligand" << endl;
else
    cout << "Receptor does not have bound ligand" << endl;
```

This snippet extracts the bound ligand from the receptor if it is present

```
OEGraphMol ligand;
if (OEReceptorHasBoundLigand(receptor))
    ligand = OEReceptorGetBoundLigand(receptor);
```

Setting a new bound ligand is done as follows

```
OEReceptorSetBoundLigand(receptor, ligand);
```

Finally a bound ligand can also be deleted from the receptor

```
OEReceptorClearBoundLigand(receptor);
```

See Also:

- `OEReceptorHasBoundLigand`
- `OEReceptorGetBoundLigand`
- `OEReceptorSetBoundLigand`
- `OEReceptorClearBoundLigand`

Extra Molecules

A receptor can contain any number of extra molecules. These molecules have no effect on docking or scoring, but can be retrieved from the receptor object or viewed in *VIDA*. This provides a mechanism for retaining structural information about water or other solvent molecules that are present in PDB files but typically must be stripped from the protein structure for docking and scoring.

This snippet sets the first extra molecule as the bound ligand of the receptor and removes it from the extra molecules list.

```

if (OEReceptorHasExtraMols (receptor))
{
    OEIter<const OEMolBase> extraMolIter = OEReceptorGetExtraMols (receptor);
    OEReceptorSetBoundLigand (receptor, *extraMolIter);

    vector<OEGraphMol> newExtraMolList;
    for (++extraMolIter ; extraMolIter ; ++extraMolIter)
        newExtraMolList.push_back (*extraMolIter);

    OEReceptorClearExtraMols (receptor);
    for (unsigned int i=0 ; i<newExtraMolList.size() ; ++i)
        OEReceptorAddExtraMol (receptor, newExtraMolList[i]);
}

```

See Also:

- [OEReceptorHasExtraMols](#)
- [OEReceptorGetExtraMols](#)
- [OEReceptorAddExtraMol](#)
- [OEReceptorClearExtraMols](#)

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.

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 protein 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.

The following code snippet checks to see what protein constraints, if any, are present

```
for (OEIter<const OEProteinConstraint> constraint
     = OEReceptorGetProteinConstraints(receptor) ;
     constraint ;
     ++constraint)
    cout << "Atom " << constraint->GetAtom()->GetIdx() << " has a constraint" << endl;
```

This snippet provides a more detailed description of each constraint

```
for (OEIter<const OEProteinConstraint> constraint
     = OEReceptorGetProteinConstraints(receptor) ;
     constraint ;
     ++constraint)
{
    cout << "Protein constraint on atom " << constraint->GetAtom()->GetIdx() << endl;
    cout << "  Type : " << OEProteinConstraintTypeGetName(constraint->GetType()) << endl;
    cout << "  Name : " << constraint->GetName() << endl;
}
```

The `OEReceptorSetProteinConstraint` function is used to set protein constraints as in the following example.

```
OEProteinConstraint proteinConstraint;
proteinConstraint.SetAtom(&*heavyAtom);
proteinConstraint.SetType(OEProteinConstraintType::Contact);
proteinConstraint.SetName("Example constraint");
OEReceptorSetProteinConstraint(receptor, proteinConstraint);
```

This example removes all protein constraints

```
OEReceptorClearProteinConstraints(receptor);
```

See Also:

- `OEReceptorHasProteinConstraints`
- `OEReceptorGetProteinConstraints`
- `OEReceptorSetProteinConstraint`
- `OEReceptorClearProteinConstraint`
- `OEReceptorClearProteinConstraints`

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.

The following code snippet checks to see if the receptor has custom constraints.

```
if (OEReceptorHasCustomConstraints(receptor))
    cout << "Receptor has custom constraints" << endl;
else
    cout << "Receptor does not have custom constraints" << endl;
```

This snippet provides a more detailed description of each constraint including what type of constraint it is, if it is enabled and its name.

```
OECustomConstraints customConstraints;
customConstraints = OEReceptorGetCustomConstraints(receptor);
for (OEIter<OEFeature> feature = customConstraints.GetFeatures();
     feature;
     ++feature)
{
    cout << "Feature " << feature->GetFeatureName() << endl;
    cout << "  Spheres : " << endl;
    for (OEIter<const OESphereBase> sphere = feature->GetSpheres();
         sphere;
         ++sphere)
    {
        cout << "    center (" << sphere->GetX();
        cout << ", " << sphere->GetY();
        cout << ", " << sphere->GetZ();
        cout << ")  radius " << sphere->GetRad() << endl;
    }
    OEIter<const string> SMARTS = feature->GetSMARTS();
    if (!SMARTS)
        cout << "  Constraint is matched by any heavy atom" << endl;
    else
    {
        cout << "  SMARTS:" << endl;
        for (; SMARTS; ++SMARTS)
            cout << "    " << *SMARTS << endl;
    }
}
}
```

This example sets a simple custom constraint that is satisfied when any heavy atom is placed within 4 Angstroms of a given protein atom.

```
string name = "Example protein contact constraint";

OECustomConstraints customConstraints;
customConstraints = OEReceptorGetCustomConstraints(receptor);

OEFeature *feature = customConstraints.AddFeature();
feature->SetFeatureName(name);

float sphereRadius = 4.0f;
```

```
float sphereCenter[3];
receptor.GetCoords(&proteinHeavyAtom, sphereCenter);
OESphereBase *sphere = feature->AddSphere();
sphere->SetRad(sphereRadius);
sphere->SetCenter(sphereCenter[0], sphereCenter[1], sphereCenter[2]);

OEReceptorSetCustomConstraints(receptor, customConstraints);
```

This example deletes all custom constraints

```
OEReceptorClearCustomConstraints(receptor);
```

See Also:

- `OEReceptorHasCustomConstraints`
- `OEReceptorGetCustomConstraints`
- `OEReceptorSetCustomConstraints`
- `OEReceptorClearCustomConstraints`

3.2 Scoring

Scoring functions in the *Docking Toolkit* measure the fitness of ligands posed within the active site of a target protein and assign them a numerical score. Poses with better scores are more likely to be correctly docked compared to other poses of the same ligand. The score of a ligand is the best score of any pose of that ligand, and ligands with better scores are more likely to be active against the target protein compared to other ligands docked.

The following scoring functions are implemented in *DockingTK*

1. *Shapegauss* (`OEScoreType::Shapegauss`)
2. *PLP* (`OEScoreType::PLP`)
3. *Chemgauss3* (`OEScoreType::Chemgauss3`)
4. *Chemgauss4* (`OEScoreType::Chemgauss4`)
5. *Chemscore* (`OEScoreType::Chemscore`)

In the *Docking Toolkit* scoring, optimization and score annotation with any of these scoring functions is done using the `OEScore` class.

Listing 1 is an example program that uses the `OEScore` class to score, optimize and annotate poses.

Required Parameters

- receptor**
Filename of receptor.
- in**
Input file of poses to rescore.
- out**
Output file for rescored molecules

Optional Parameters

- score**
Scoring function to use (defaults to `OEScoreType::Default`)

-optimize

Flag to optimize molecules before rescoring

Listing 1: Example program for rescoring, optimizing and annotating molecules

```
#include "openeye.h"
#include "oesystem.h"
#include "oechem.h"
#include "oedocking.h"
#include <string>

using namespace OESystem;
using namespace OEChem;
using namespace OEDocking;
using namespace std;

#include "RescorePoses.itf"

int main(int argc, char** argv)
{
    OEInterface itf(InterfaceData);
    OEScoreTypeConfigure(itf, "-score");
    if (!OEParseCommandLine(itf, argc, argv))
        return 1;

    oemolistream imstr(itf.Get<string>("-in"));
    oemolostream omstr(itf.Get<string>("-out"));

    OEGraphMol receptor;
    if (!OEReadReceptorFile(receptor, itf.Get<string>("-receptor")))
        OEThrow.Fatal("Unable to read receptor");

    unsigned int scoreType = OEScoreTypeGetValue(itf, "-score");
    OEScore score(scoreType);
    score.Initialize(receptor);

    OEMol ligand;
    bool optimize = itf.Get<bool>("-optimize");
    while (OEReadMolecule(imstr, ligand))
    {
        if (optimize)
            score.SystematicSolidBodyOptimize(ligand);
        score.AnnotatePose(ligand);
        string sdtag = score.GetName();
        OESetSDScore(ligand, score, sdtag);
        OESortConfsBySDTag(ligand, sdtag, score.GetHighScoresAreBetter());
        OEWwriteMolecule(omstr, ligand);
    }
    return 0;
}
```

3.2.1 Choosing a scoring function

The scoring function is chosen when the `OEScore` object is constructed, as shown in this snippet of code from Listing 1.

```
OEScore score(scoreType);
```

scoreType is an unsigned int constant from the `OEScoreType` namespace identifying a scoring function.

Supported scoring functions are

- *Shapegauss* (`OEScoreType::Shapegauss`)
- *PLP* (`OEScoreType::PLP`)
- *Chemscore* (`OEScoreType::Chemscore`)
- *Chemgauss3* (`OEScoreType::Chemgauss3`)
- *Chemgauss4* (`OEScoreType::Chemgauss4`)

Note: `OEScore` may also be constructed without the *scoreType* parameter, in which case the default scoring function (*Chemgauss4*) is used.

3.2.2 Initialization

An `OEScore` object must be initialized with the structure of the target protein and the location of the active site. This is done by passing either a receptor object (see *Receptors* chapter) or protein and box enclosing the active site to the `OEScore::Initialize` method.

Initializing with a receptor is the simplest method as shown in this snippet of code from [Listing 1](#).

```
score.Initialize(receptor);
```

`OEScore` objects can also be initialized using a protein and a box enclosing the active site. The following example initializes an `OEScore` object using the structure of a target protein and a box that is setup by first enclosing a docked ligand and then adding a 4 Angstrom padding to all sides of the box.

```
OEGraphMol protein;
oemolistream imstr(argv[1]);
OEReadMolecule(imstr, protein);

OEGraphMol pose;
imstr.open(argv[2]);
OEReadMolecule(imstr, pose);

float addbox = 4.0f;
OEBox box;
OESetupBox(box, pose, addbox);

OEScore oescore;
oescore.Initialize(protein, box);
```

Initialization can take 1-2 minutes for large active sites. This time is invariant to the initialization method (*i.e.* receptor vs. protein-box).

See Also:

- `OEScore::Initialize`

3.2.3 Scoring Poses

The `OEScore` class can calculate the score of a pose as well as break down that score into contributions from individual components of the scoring function as shown in the following example.

```
void PrintScore(OEScore& score, const OEMolBase& pose)
{
    cout << "Total ligand score = " << score.ScoreLigand(pose) << endl;

    cout << "Score components contributions to score:" << endl;
    for (OEIter<const string> comp = score.GetComponentNames(); comp; ++comp)
        cout << *comp << ": " << score.ScoreLigandComponent(pose, *comp) << endl;
}
```

Scores can also be calculated on a per-atom basis, and broken down into contributions from individual components of the scoring function.

```
void PrintAtomScore(OEScore& score,
                  const OEMolBase& pose,
                  const OEAtomBase* atom)
{
    cout << endl << "Atom: " << atom->GetIdx() << " score:" << score.ScoreAtom(atom, pose) << endl;

    cout << "Score components contribution to atom scores:" << endl;
    for (OEIter<const string> comp = score.GetComponentNames(); comp; ++comp)
        cout << *comp << ": " << score.ScoreAtomComponent(atom, pose, *comp) << endl;
}
```

Listing 1 uses the convenience function `OESetSDScore` to assign the scores to SD data on the molecules. This function simply assigns the result of `OEScore::ScoreLigand` to SD data with tag *sdtag*.

```
OESetSDScore(ligand, score, sdtag);
```

See Also:

- `OEScore::ScoreLigand`
- `OEScore::ScoreLigandComponent`
- `OEScore::ScoreAtom`
- `OEScore::ScoreAtomComponent`
- `OESetSDScore`

3.2.4 Annotating

Annotating is a method of attaching score information to a molecule that can then be viewed in *VIDA*. When an annotated molecule is loaded into *VIDA* the score information will appear in the list window and the scores of individual atoms shown in the 3D window. The following snippet of code from Listing 1 annotates a pose.

```
score.AnnotatePose(ligand);
```

Warning: Annotated poses must be saved in `.oeb` or `.oeb.gz` format. Saving to another format will result in the loss of the annotation information.

See Also:

```
OEScore::AnnotatePose
```

3.2.5 Optimizing

The `OEScore` class can optimize poses using a systematic solid body optimization, as shown in this snippet from Listing 1

```
score.SystematicSolidBodyOptimize(ligand);
```

This optimization is rigid with respect to the ligand and the protein. Each pose is optimized by taking one positive and one negative step for each of the 6 degrees of freedom (3 rotational and 3 translational) for a total of 729 positions tested (3 to the 6th).

See Also:

```
OEScore::SystematicSolidBodyOptimize
```

3.2.6 Sorting Poses

Multiple poses of a docked molecule are generally stored as conformers of an `OEMol` object. The original ordering of poses in the `OEMol` is not altered when `OEScore::SystematicSolidBodyOptimize` is called (or any of the score assignment free functions). Listing 1 uses **OEChem** free function `OESortConfsBySDTag` to insure that the rescored poses of each molecule are ordered from best to worst.

```
OESortConfsBySDTag(ligand, sdtag, score.GetHighScoresAreBetter());
```

The method `OEScore::GetHighScoresAreBetter` will return false if lower value scores are better scores and true if higher value scores are better scores.

See Also:

```
OESortConfsBySDTag
```

3.2.7 Command Line Interface

Listing 1 uses the `OEInterface` class to provide its command line interface. The primary documentation for the `OEInterface` class is in the *OEChem* documentation. Listing 1, however, makes uses of 2 convenience functions unique to the *Docking Toolkit* to define and use the `-score` command line parameter.

The following snippet from the example program shown in Listing 1 adds the `-score` flag to the command line, which allows the user to set the scoring function (see *Scoring Function Implementations* section).

```
OEScoreTypeConfigure(itf, "-score");
```

The user specified value, or the default value `OEScoreType::Default`, is obtained in the following snippet from Listing 1.

```
unsigned int scoreType = OEScoreTypeGetValue(itf, "-score");
```

See Also:

- `OEScoreTypeConfigure`
- `OEScoreTypeGetValue`

3.2.8 Scoring Function Implementations

Shapegauss

Shapegauss [McGann-2003] is a shape based scoring function that favors poses that complement the active site well, regardless of any chemical interactions (*e.g.* hydrogen bonds). The Shapegauss scoring function represents the shape of each atom as a smooth Gaussian function.

The Shapegauss score is calculated by summing a pairwise potential between all protein atoms and all ligand heavy atoms. This potential is most favorable when the two atoms touch but do not overlap. A correction term is then applied to further penalize atoms which significantly overlap the protein.

PLP

PLP [Verkivker-2000] or Piecewise Linear Potential scoring function calculates both the shape and hydrogen bond complementarity of poses to the active site.

The PLP score is a pairwise additive scoring function. All pairs of ligand-protein heavy atoms are assigned either a hydrogen bonding potential, if the pair is an acceptor-donor pair, or otherwise a lipophilic potential. These pairwise potentials are summed to obtain the final pose score.

The PLP implementation in the *Docking Toolkit* has also been extended to include interaction between metals on the target protein and acceptors on the ligand.

Chemscore

Chemscore [Eldridge-1997] is a sum of the following interaction terms

lipophilic

Favorable interactions that occur when two non-polar heavy atoms (one ligand atom and one protein atom) are placed near each other.

hydrogen bonding

Favorable interactions that occur when acceptor-donor interactions are formed between the ligand and protein.

metal chelator

Favorable interactions that occur when acceptor atoms on the ligand are placed near metal atoms on the protein.

clash

Penalty term that occurs when heavy atoms on the ligand and protein clash.

rotatable bond

This penalty term is proportional to the number of rotatable bonds on the ligand that are no longer free to rotate when the ligand is docked.

Chemgauss3

The Chemgauss3 scoring function uses Gaussian smoothed potentials to measure the complementarity of ligand poses within the active site. Chemgauss3 recognizes the following types of interactions.

- Shape

- Hydrogen bonding between ligand and protein
- Hydrogen bonding interactions with implicit solvent
- Metal-chelator interactions.

All interaction potentials in Chemgauss are initially constructed using step functions to describe the interaction of atom pairs (or other chemical points) as a function of distance. These interactions are mapped onto a grid that is then convoluted with a spherical Gaussian function, which smoothes the potential making it less sensitive to small changes in the ligand position. Smoothing the score in this way serves two purposes, first docking can be run at lower resolution than would be required if the score were not smooth since small changes in position do not cause large changes in score. Second it reduces the error associated with the rigid protein approximation by effectively accounting for the ability of the protein to make small structural re-arrangements to accommodate the ligand.

Shape interactions in Chemgauss are based on a united atom model (*i.e.* only heavy atoms are relevant to the shape calculation). Each ligand heavy atom is assigned a fixed clash penalty score if the distance between it and a protein heavy atom is less than the sum of the VdW radii, otherwise it is assigned a score proportional to the count of the number of protein heavy atoms within 1.25 and 2.5 times the sum of the VdW radii (atoms within 2.5 count one tenth as much as those within 1.25). From this score a penalty equal to two close protein atom contacts is subtracted to represent the VdW interactions with solvent water that are lost when the ligand docks. This score is pre-computed at grid points throughout the active site and the resulting grid is then smoothed.

Hydrogen bonding groups are modeled with one or more lone-pair or polar-hydrogen positions that describe the directionality of potential hydrogen bonds (with respect to the hydrogen bonding group's heavy atom). Donor groups have lone pair positions representing the possible location of the donor hydrogen atoms relative to the donating molecule, while acceptors have lone-pair positions representing the possible locations of the donated hydrogen relative to the acceptor. A hydrogen bond is detected and assigned a constant score when a hydrogen bonding position on the ligand is within 1.0 Angstroms of a complementary hydrogen bonding position on the protein (*i.e.* when the polar-hydrogen position of a donor overlaps the lone-pair position of an acceptor). If the ligand hydrogen bonding group has multiple polar-hydrogens and/or lone-pair positions (groups can be both donors and acceptors) then this calculation is performed for each position and the result is summed. As with all Chemgauss terms the hydrogen bond potential is pre-computed at grid points throughout the site and then smoothed.

Hydrogen bonds with solvent that break when the ligand docks into the active site are penalized by the Chemgauss scoring function. Broken protein-solvent hydrogen bonds are accounted for by calculating how many hydrogen bonds water can make with the protein at the position of each heavy atom of the docked ligand, and a penalty score is assigned which is proportional to the number of hydrogen bonds. Broken ligand-solvent hydrogen bonds are accounted for by calculating desolvation positions around each hydrogen-bonding group on the ligand that represent the positions water could occupy when making a hydrogen bonding interaction with the protein. A penalty is then assessed that is proportional to the number of desolvation positions that can no longer be occupied by water because the water in these positions would clash with the protein. As before, this potential is placed on a grid and smoothed.

Chelating interactions between protein metals and ligand chelating groups are accounted for by Chemgauss (protein-chelator and ligand-metal chelating interactions are not). For each chelator on the ligand one or more chelating-positions are calculated. If a protein metal is within 1.0 Angstroms of any chelating-position of a chelating group then a fixed score is assigned, otherwise a zero score is assigned. As before this potential is placed on a grid and smoothed.

Chemgauss4

The Chemgauss4 is a modification of the Chemgauss3 scoring function that has improved hydrogen bonding and metal chelator (The shape and implicit solvent interaction terms are identical to those in Chemgauss3). The new hydrogen bonding and metal chelator terms have better perception of the directionality of these interactions and also account for hydrogen bond networking effects.

To calculate the hydrogen bonding score for a ligand-protein hydrogen bond two distances are measured.

1. How far the donor heavy atom is from the position the acceptor atom would consider to be an ideal for a hydrogen bonding to form.
2. How far the acceptor heavy atom is from the position the donor atom would consider to be ideal for a hydrogen bonding interaction to occur.

The score for the hydrogen bond interaction is a product of two Gaussian functions of these distances scaled by the strength of the hydrogen bonding groups involved in the interaction.

$$\text{HBscore} = \text{strength} * g(\text{distance1}) * g(\text{distance2})$$

To compute the total hydrogen bonding score for the ligand-protein complex the individual pairwise scores are calculated for all protein-ligand donor-acceptor. Individual HB interaction are then eliminated if either the donor or acceptor exceeds the maximum number of interactions allowed (e.g., a hydroxyl with one hydrogen is not allowed to make more than one donor interaction), with the lowest scoring interactions eliminated first. The final hydrogen bond score is then calculated by summing the scores of the remained individual acceptor-donor interactions.

Chemical Gaussian Overlay

The Chemical Gaussian Overlay function (or CGO) is a hybrid scoring function that scores poses based on their similarity to a known bound ligand and the interactions both the docked and bound ligand make with the protein active site. **This scoring function is not implemented by the `OEScore` class.** This scoring function is used by the `OEDock` class during the exhaustive search (see *Docking Algorithm* section) when using the hybrid docking method (see *Hybrid Method* section).

CGO is primarily a ligand-based scoring functions although some information from the protein structure is used as well. The similarities computed are based on the overall shape of the molecules as well as the position of hydrogen bonding and metal chelating groups. This scoring function requires a bound ligand pose along with the structure of the target protein. Typically the ligand structure is obtained from X-ray crystallography along with the structure of the target protein, although a docked ligand could also, in principal, be used.

CGO represents molecules as a set of spherical Gaussian functions describing the shape and chemistry (acceptors, donors and chelators) of the molecule. The Gaussians representing the shape of the molecule are centered at the heavy atom positions, those for donors are centered on polar-hydrogen positions (*i.e.* positions where the donating hydrogen could be when it is involved in a hydrogen bond), those for acceptors are centered on lone-pair positions (*i.e.* positions where a donating hydrogen could be when a hydrogen bond is formed) and those for chelators are centered at chelating positions (*i.e.* locations where a metal could have a chelating interaction). The overlap of the Gaussians on the docked ligand to those on the bound ligand are computed for each type of Gaussian (*e.g.* shape, donor, acceptor and chelator) by summing the overlap of individual pairs of Gaussian. The overlap of each individual pair is calculated by integrating the product of the two. To prevent chemistry not relevant to binding from contributing to the overall score, when calculating the chemistry overlaps (*i.e.* acceptor, donor and chelator) only groups that make the interaction with the protein are accounted for (*e.g.* a chelator that does not interact with a metal on the protein is ignored in the overlap calculation). The sum of all four types of overlaps is the CGO score.

3.3 Docking

Docking is the process of determining the structure of a ligand bound in the active site of a target protein. In the *Docking Toolkit* this is done with the `OEDock` class that takes a multiconformer representation of a ligand and returns the top scoring pose (or poses if desired) within the active site. Docking is done using an exhaustive search algorithm, followed by optimization of the best poses from the exhaustive search (see *Docking Algorithm* section).

Listing 2 is an example program that uses the `OEDock` class to dock, score and annotate multiconformer molecules.

Required Parameters

- receptor**
Filename of a receptor.
- in**
Input file of multiconformer molecules to dock.
- out**
Output file for docked molecules.

Optional Parameters

- method**
Scoring method to use.
- resolution**
Docking resolution to use.

Listing 2: Example program for docking molecules

```

#include "openeye.h"
#include "oesystem.h"
#include "oechem.h"
#include "oedocking.h"

using namespace OESystem;
using namespace OEChem;
using namespace OEDocking;
using namespace std;

#include "DockMolecules.itf"

int main(int argc, char** argv)
{
    OEInterface itf(InterfaceData);
    OEDockMethodConfigure(itf, "-method");
    OESearchResolutionConfigure(itf, "-resolution");
    if (!OEParseCommandLine(itf, argc, argv))
        return 1;

    oemolistream imstr(itf.Get<string>("-in"));
    oemolostream omstr(itf.Get<string>("-out"));

    OEGraphMol receptor;
    if (!OEReadReceptorFile(receptor, itf.Get<string>("-receptor")))
        OThrow.Fatal("Unable to read receptor");

    unsigned int dockMethod = OEDockMethodGetValue(itf, "-method");
    unsigned int dockResolution = OESearchResolutionGetValue(itf, "-resolution");
    OEDock dock(dockMethod, dockResolution);
    dock.Initialize(receptor);

    OEMol mcmol;
    while (OEReadMolecule(imstr, mcmol))
    {
        OEGraphMol dockedMol;
        dock.DockMultiConformerMolecule(dockedMol, mcmol);
        string sdtag = OEDockMethodGetName(dockMethod);
        OESetSDScore(dockedMol, dock, sdtag);
    }
}

```

```

    dock.AnnotatePose(dockedMol);
    OEWriteMolecule(omstr, dockedMol);
  }
  return 0;
}

```

3.3.1 Scoring Functions and Search Resolution

The resolution of the exhaustive search and scoring functions used are chosen when the `OEDock` object is constructed, as shown in this snippet of code from [Listing 2](#).

```
OEDock dock(dockMethod, dockResolution);
```

`dockMethod` is an unsigned int constant from the `OEDockMethod` namespace, that specifies the combination of scoring functions (or method) `OEDock` uses for the exhaustive search and optimization. The available scoring methods are:

| Method | Exhaustive Search Scoring | Optimization Scoring |
|---------------------------------------|----------------------------------|----------------------|
| <code>OEDockMethod::Shapegauss</code> | <i>Shapegauss</i> | <i>Shapegauss</i> |
| <code>OEDockMethod::PLP</code> | <i>PLP</i> | <i>PLP</i> |
| <code>OEDockMethod::Chemgauss3</code> | <i>Chemgauss3</i> | <i>Chemgauss3</i> |
| <code>OEDockMethod::Chemgauss4</code> | <i>Chemgauss3</i> | <i>Chemgauss4</i> |
| <code>OEDockMethod::Chemscore</code> | <i>Chemgauss3</i> | <i>Chemscore</i> |
| <code>OEDockMethod::Hybrid</code> | <i>Chemical Gaussian Overlay</i> | <i>Chemgauss3</i> |
| <code>OEDockMethod::Hybrid</code> | <i>Chemical Gaussian Overlay</i> | <i>Chemgauss4</i> |

`dockResolution` is an unsigned int constant from the `OESearchResolution` namespace, that specifies the docking resolution to use. The docking resolution is the rotational and translational stepsize used during the exhaustive search and optimization (the rotational stepsize is the furthest distance any heavy atom of the ligand will move in a single rotational step). The following resolutions are supported

| Resolution | Exhaustive Translational | Exhaustive Rotational |
|---|--------------------------|-----------------------|
| <code>OESearchResolution::High</code> | 1.0 | 1.0 |
| <code>OESearchResolution::Standard</code> | 1.0 | 1.5 |
| <code>OESearchResolution::Low</code> | 1.5 | 2.0 |

| Resolution | Optimization Translational | Optimization Rotational |
|---|----------------------------|-------------------------|
| <code>OESearchResolution::High</code> | 0.5 | 0.5 |
| <code>OESearchResolution::Standard</code> | 0.5 | 0.75 |
| <code>OESearchResolution::Low</code> | 0.75 | 1.0 |

All resolutions are in Angstroms.

Note: The `OEDock` constructor has default settings for both `dockMethod` and `dockResolution`. The default value are `OEDockMethod::Chemgauss4` and `OESearchResolution::Standard` respectively.

See Also:

Section *Docking Algorithm* for more information docking algorithm.

Hybrid Method

The hybrid (`OEDockMethod::Hybrid2`) docking method is distinguished from other docking methods because it uses information present in the structure of a bound ligand to enhance docking performance.

The bound ligand information is used during the exhaustive search stage of the docking process (see *Docking Algorithm* section) by using the *Chemical Gaussian Overlay* scoring function, which scores poses based on how well a docked pose overlays the shape of the bound ligand and mimics the same hydrogen bonding and interactions the bound ligand makes. After the exhaustive search, optimization is performed with *Chemgauss4* which is a standard structure based scoring function.

The final score a ligand receives using the hybrid docking method is based only on its interactions with the protein. Ligand information is used only to guide the selection of poses during the exhaustive search.

Note: In order to use the hybrid docking method the receptor object must have a bound ligand (see *Bound Ligand* section).

3.3.2 Initialization

An `OEDock` object must be initialized with a receptor object, prior to docking, scoring or annotating any molecules. This is done by passing a receptor to the `OEDock::Initialize` method, as shown in the following code snippet from [Listing 2](#).

```
dock.Initialize(receptor);
```

Note: Unlike `OEScore`, `OEDock` **cannot** be initialized with a protein and a box, however a receptor can be constructed from a protein and box using the `OEMakeReceptor` function.

See Also:

- `OEDock::Initialize`

3.3.3 Docking Molecules

Once the `OEDock` object has been initialized molecules are docked using the `OEDock::DockMultiConformerMolecule` method, as shown in this code snippet from [Listing 2](#).

```
dock.DockMultiConformerMolecule(dockedMol, mcmol);
```

`mcmol` is a multiconformer representation of the molecule being docked, and `pose` is an `OEMolBase` the resulting top scoring docked pose is returned in. The score of the docked molecule can be obtained by calling the `OEMolBase::GetEnergy` method of `pose`.

`OEDock` can also return alternate as well as top scoring poses of the docked molecule.

```
unsigned int numPoses = 10;
OEMol poses;
dock.DockMultiConformerMolecule(poses, mcmol, numPoses);
```

In this example the 10 best scoring poses are returned as conformers of `poses`. The score of each pose can be obtained by calling the `OEMolBase::GetEnergy` method of each pose.

See Also:

- `OEDock::DockMultiConformerMolecule`

3.3.4 Scoring Molecules

The final score of a molecule (using the optimization scoring function) docked with the `OEDock::DockMultiConformerMolecule` method can be obtained by calling the `OEConfBase::GetEnergy`.

`OEDock` can also recalculate a score of a pose (using the optimization scoring function), and calculate the contribution for each individual component of the score as in the following example.

```
void PrintScore(OEDock& dock, const OEMolBase& pose)
{
    cout << "Total pose score = " << dock.ScoreLigand(pose) << endl;

    cout << "Score components contributions to score:" << endl;
    for (OEIter<const string> comp = dock.GetComponentNames() ; comp ; ++comp)
        cout << *comp << ": " << dock.ScoreLigandComponent(pose, *comp) << endl;
}
```

Scores can also be calculated on a per-atom basis, and broken down into contributions from individual components of the scoring function.

```
void PrintAtomScore(OEDock& dock,
                  const OEMolBase& pose,
                  const OEAtomBase* atom)
{
    cout << endl << "Atom: " << atom->GetIdx() << " score: " << dock.ScoreAtom(atom, pose) << endl;

    cout << "Score components contributions to atoms score: " << endl;
    for (OEIter<const string> comp = dock.GetComponentNames() ; comp ; ++comp)
        cout << *comp << ": " << dock.ScoreAtomComponent(atom, pose, *comp) << endl;
}
```

[Listing 2](#) uses the convenience function `OESetSDScore` to assign the scores to SD data on the molecules. This function simply assigns the result of `OEScore::ScoreLigand` to the SD data with tag `sdtag`.

```
OESetSDScore(dockedMol, dock, sdtag);
```

See Also:

- `OEDock::ScoreLigand`
- `OEDock::ScoreLigandComponent`
- `OEDock::ScoreAtom`
- `OEDock::ScoreAtomComponent`
- `OESetSDScore`

3.3.5 Annotating

`OEDock` can add *VIDA* score annotations to docked poses, as shown in this code snippet from [Listing 2](#).

```
dock.AnnotatePose(dockedMol);
```

These annotations break down the score of each pose to contributions from each component of the scoring function and each atom of the scoring function. When viewed in *VIDA* these values are displayed visually in the 3D window.

See Also:

- `OEDock::AnnotatePose`

3.3.6 Docking Algorithm

`OEDock` docks multiconformer molecules using an exhaustive search that systematically searches rotations and translations of each conformer of the ligand within the active site. Following the exhaustive search the top scoring poses are optimized and assigned a final score. These two steps are described in more detail below

Exhaustive Search

1. Enumerates, to given resolution, every possible rotation and translation of each conformer of the ligand being docked within a box enclosing the active site. The resolution of the exhaustive search is determined by the overall resolution setting of `OEDock` (see *Scoring Functions and Search Resolution* section).
2. Discard out poses that either clash with the protein or extend to far from the binding site using the receptor's negative image outer contour (see *Negative Image* section).
3. If the negative image inner contour is enabled discard any poses that do not have at least one heavy atom that falls within the inner contour. (see *Negative Image* section).
4. Discard any poses that do no match any user specified constraints. (see *Constraints* section).
5. Score all remaining poses. The scoring function used here depends on what scoring method is being used (see *Scoring Functions and Search Resolution* section).
6. Sort poses by score and pass the top scoring poses to optimization. The number of poses passed to optimization depends on the search resolution (see `OESearchResolution` constant namespace).

Optimization

1. Each pose moved to 729 nearby positions, scored and the top scoring position become the new optimized pose. The positions are generated by having the initial pose take one positive and one negative step for each translational and rotational degree of freedom. The resolution of these steps is half that of the exhaustive search, and is determined by the overall resolution setting of `OEDock` (see *Scoring Functions and Search Resolution* section).
2. The best scoring pose of the 729 tests poses is selected as the final docked structure (and score) for the ligand.

4.1 OEDocking Classes

4.1.1 OEBoxBase

class OEBoxBase

OEBoxBase class represents a box aligned to the x, y and z coordinate axis.

This pure virtual class is implemented by OEBox.

The following free functions take OEBoxBase.

| | |
|-------------------|----------------|
| OEBoxXMid | OEBoxXDim |
| OEBoxYMid | OEBoxYDim |
| OEBoxZMid | OEBoxZDim |
| OEInBox | OEBoxTranslate |
| OEBoxExtend | OESetupBox |
| OEBoxVolume | OEBoxArea |
| OEMakeBoxMolecule | |

operator=

OEBoxBase& **operator=**(const OEBoxBase& rhs)

Assignment operator to another OEBoxBase

Destructor

virtual ~OEBoxBase() {}

Destructor

GetXMax

virtual float GetXMax() const = 0

Returns the maximum X value of the box

GetYMax

```
virtual float GetYMax() const = 0
```

Returns the maximum Y value of the box

GetZMax

```
virtual float GetZMax() const = 0
```

Returns the maximum Z value of the box

GetXMin

```
virtual float GetXMin() const = 0
```

Returns the minimum X value of the box

GetYMin

```
virtual float GetYMin() const = 0
```

Returns the minimum Y value of the box

GetZMin

```
virtual float GetZMin() const = 0
```

Returns the minimum z value of the box

Setup

```
virtual bool Setup(float x1,  
                  float y1,  
                  float z1,  
                  float x2,  
                  float y2,  
                  float z2) = 0
```

Sets up the box. 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.

4.1.2 OEBox

```
class OEBox : public OEBoxBase
```

OEBox class represents a box aligned to the x, y and z coordinate axis.

This class is an implementation of the OEBoxBase class.

The following free functions can take OEBox objects by virtue of their inheritance from OEBoxBase.

| | |
|-------------------|----------------|
| OEBoxXMid | OEBoxXDim |
| OEBoxYMid | OEBoxYDim |
| OEBoxZMid | OEBoxZDim |
| OEInBox | OEBoxTranslate |
| OEBoxExtend | OESetupBox |
| OEBoxVolume | OEBoxArea |
| OEMakeBoxMolecule | |

Constructors

OEBox ()

Default constructor. All xyz min and max values are set to 0.

```
OEBox(float x1,
      float y1,
      float z1,
      float x2,
      float y2,
      float z2)
```

Constructor that sets up the box. The max and min X values will be set to the max and min of $x1$, $x2$ respectively. Minimum and maximum values of Y and Z are selected in an analogous fashion.

```
OEBox(const OEChem::OEMolBase& mol, float addbox = 0.0f)
```

Constructs a box around the given molecule. The maximum and minimum x, y and z values are set to the maximum and minimum values x, y and z coordinates of any atom of the molecule.

If *addbox* is specified then each edge of the box is extended by *addbox*.

```
OEBox(const OEChem::OEMCMolBase& mol, float addbox = 0.0f)
```

Constructs a box around the given molecule. The maximum and minimum x, y and z values are set to the maximum and minimum values x, y and z coordinates of any atom of any pose of the molecule.

If *addbox* is specified then each edge of the box is extended by *addbox*.

```
OEBox(const OEBox& box)
```

Copy constructor.

```
OEBox(const OEBoxBase& box)
```

Constructs an OEBox from an OEBoxBase.

operator=

```
OEBox& operator=(const OEBox& rhs)
```

Assignment operator

```
OEBox& operator=(const OEBoxBase& rhs)
```

Assignment operator for `OEBoxBase`.

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

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

Setup

```
bool Setup(float x1,
           float y1,
           float z1,
           float x2,
           float y2,
           float z2)
```

Sets up the box. 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.

4.1.3 OECustomConstraint

```
class OECustomConstraint
```

`OECustomConstraint` holds a set of custom docking constraints used by `OEDock`. Each custom constraint is defined by an `OEFeature` contained by this class.

To use the constraints specified by an `OECustomConstraint` object the object must be passed to the receptor with `OEReceptorSetCustomConstraints` prior to calling the `OEDock::Initialize` method of `OEDock`.

Constructors

```
OECustomConstraint()
```

Default constructor. Object contains no features in the default constructed state.

```
OECustomConstraint(const OECustomConstraint& )
```

Copy Constructor.

operator=

```
OECustomConstraint& operator=(const OECustomConstraint&)
```

Assignment operator.

GetFeatures

```
OESystem::OEIterBase< OEFeature>* GetFeatures(bool enabledOnly = true)
OESystem::OEIterBase<const OEFeature>* GetFeatures(bool enabledOnly = true) const
```

Returns an iterator over all features contained by this class, or all enabled features if *enabledOnly* is true (see `OEFeature::GetEnabled` method).

AddFeature

```
OFeature* AddFeature()
```

Adds a new `OFeature` object to this class and returns a pointer to the newly created `OFeature`. The returned `OFeature` object is owned by this class, and will be destroyed by the destructor of this class.

DeleteFeature

```
bool DeleteFeature(const OFeature* feature)
```

Deletes an `OFeature` owned by this class. Note that after this method is called *feature* will no longer point to a valid `OFeature` object.

NumFeatures

```
unsigned int NumFeatures(bool enabledOnly = true) const
```

Returns the number of `OFeature` objects contained by this class.

If *enabledOnly* is true then only `OFeature` objects for which the method `OFeature::GetEnabled` returns true will be counted.

Clear

```
bool Clear()
```

Returns this object to its default constructed state, deleting any `OFeature` objects it may currently be holding.

4.1.4 OEDock

```
class OEDock
```

`OEDock` is used to dock and score multiconformer molecules in an active site.

Use the following procedure to dock and score molecules with this class.

1. Select the scoring method and docking search resolution at construction time.

To select a scoring method pass one of the following constants as the first argument of the constructor:

- `OEDockMethod::Shapegauss`
- `OEDockMethod::PLP`
- `OEDockMethod::Chemscore`
- `OEDockMethod::Chemgauss3`
- `OEDockMethod::Chemgauss4`
- `OEDockMethod::Chemgauss`
- `OEDockMethod::Hybrid1`

- `OEDockMethod::Hybrid2`
- `OEDockMethod::Hybrid`
- `OEDockMethod::Default`

To select a search resolution pass one of the following constraints as the section argument of the constructor

- `OESearchResolution::High`
- `OESearchResolution::Standard`
- `OESearchResolution::Low`

2. Initialize the active site by passing a receptor (see *Receptors* chapter) to the `OEDock::Initialize` method.
3. Dock molecules using the `OEDock::DockMultiConformerMolecule` method.
4. Score docked molecules with any of the following methods
 - `OEDock::ScoreLigand`
 - `OEDock::ScoreAtom`
 - `OEDock::ScoreLigandComponent`
 - `OEDock::ScoreAtomComponent`
 - `OEDock::AnnotatePose`

Constructor

```
OEDock(unsigned int scoring = OEDockMethod::Default,
        unsigned int resolution = OESearchResolution::Default)
```

Constructs the OEDock object to use a given *scoring* method and search *resolution*.

Table 4.1: Parameters

| | |
|------------|--|
| scoring | The scoring method, from the <code>OEDockMethod</code> namespace, this object will use. |
| resolution | The search resolution, from the <code>OESearchResolution</code> namespace, this object will use. |

IsInitialized

```
bool IsInitialized() const
```

Returns true if this object has been successfully initialized and is ready to dock molecules.

Initialize

```
bool Initialize(const OEChem::OEMolBase& receptor)
```

Sets up this object to dock molecules into a receptor.

This function returns true if setup was successful and false otherwise.

receptor A receptor describing the active site of the target protein (see *Receptors* chapter). This object does not depend upon receptor once this call is completed (*i.e.* receptor can be destroyed after calling this method).

Note: This method can take several minutes to complete with larger active sites.

DockMultiConformerMolecule

Both overloads of this method dock *inputMol* into the receptor passed to the `OE Dock::Initialize` method.

The return value of this method describes the result of the docking, with a value from the `OE DockingReturnCode` namespace. A result of `OE DockingReturnCode::Success` indicates docking was successful.

```
unsigned int DockMultiConformerMolecule(OEChem::OEMolBase& dockedMol,
                                         const OEChem::OEMCMolBase& inputMol)
```

This overload of `OE Dock::DockMultiConformerMolecule` returns the top scoring pose.

dockedMol Top scoring docked pose of *inputMol*.

inputMol A multiconformer representation of a molecule to dock.

The score of the docked pose can be obtained by calling the `OEMolBase::GetEnergy` method of *dockedMol*.

```
unsigned int DockMultiConformerMolecule(OEChem::OEMCMolBase& dockedMol,
                                         const OEChem::OEMCMolBase& inputMol,
                                         unsigned int numPoses = 1)
```

This overload of `OE Dock::DockMultiConformerMolecule` can return alternate docked poses, in addition to the top scoring pose.

dockedMol Docked poses of *inputMol*. Poses are stored as conformers of the `OEChem::OEMCMolBase` and are sorted score.

inputMol A multiconformer representation of a molecule to dock.

numPoses Maximum number of top scoring docked poses to return in *dockedMol*. Typically this will be the number of poses returned, however, in highly restricted sites fewer than *numPoses* may be returned. The value of *numPoses* must be greater than zero.

The score of the docked poses can be obtained by calling the `OEMolBase::GetEnergy` method on the conformers of *dockedMol*.

GetHighScoresAreBetter

```
bool GetHighScoresAreBetter() const
```

Returns true if higher scores indicate a better result.

Returns false if lower scores indicate a better result.

GetName

```
std::string GetName() const ;
```

Returns the name of the scoring function docked poses are scored with.

GetComponentNames

```
OESystem::OEIterBase<const std::string>* GetComponentNames() const
```

ScoreLigand

```
float ScoreLigand(const OEChem::OEMolBase& pose)
```

Rescores a pose within the active site.

pose Structure of a pose within the active site

If an error occurs this function will return FLT_MAX if `OE Dock::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

ScoreAtom

```
float ScoreAtom(const OEChem::OEAtomBase& atom,
                const OEChem::OEMolBase& pose)
```

Returns the score of an atom of a given pose within the active site.

atom Atom of *pose* to score.

pose Structure of a pose within the active site

If an error occurs this function will return FLT_MAX if `OE Dock::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

ScoreLigandComponent

```
float ScoreLigandComponent(const OEChem::OEMolBase& pose,
                           std::string compName)
```

Returns the given components contribution to the total score.

pose Structure of a pose within the active site

compName Name of the score component to report. Name must be one returned by `OE Dock::GetComponentNames`.

If an error occurs this function will return FLT_MAX if `OE Dock::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

ScoreAtomComponent

```
float ScoreAtomComponent(const OEChem::OEAtomBase& atom,
                        const OEChem::OEMolBase& pose,
                        std::string compName)
```

Returns the given components contribution to the score of a given atom score.

atom Atom of *pose* to score.

pose Structure of a pose within the active site

compName Name of the score component to report. Name must be one returned by `OE Dock::GetComponentNames`.

If an error occurs this function will return `FLT_MAX` if `OE Dock::GetHighScoresAreBetter` returns false or `-FLT_MAX` otherwise.

AnnotatePose

Adds **VIDA** scoring annotation to the *pose* or *poses* passed.

The annotated poses must be written out in either *oeb* or *oeb.gz* format and are only viewable in **VIDA**.

bool `AnnotatePose(OEChem::OEMolBase& pose)`

This overload of `OE Dock::AnnotatePose` annotates a single pose.

pose Structure of a pose within the active site

bool `AnnotatePose(OEChem::OEMCMolBase& poses)`

This overload of `OE Dock::AnnotatePose` annotates all poses of a given ligand.

poses An `OEMCMolBase` the conformers of which are poses within the active site.

CacheScoringSetup

This function caches the current scoring setup of the `OE Dock` object onto a receptor object. When another `OE Dock` object is initialized with this receptor the it will read in the cached score data rather than recalculating it from scratch, thus improving the startup time of the `OE Dock` object.

bool `CacheScoringSetup(OEChem::OEMolBase& receptor, bool clearOldData = true)`

receptor A receptor object. This must be the same receptor object the `OE Hybrid` object was initialized with or an exact copy.

clearOldData Flag to clear cached data from a prior call of `CacheScoringSetup` with this receptor. The cached data can be quite sizeable (hundreds of megabytes), so leaving this flag at the default value of true is recommended.

Note: The cached score data on a receptor will be saved when the receptor is written to a file.

4.1.5 OEFeature

class `OEFeature`

`OEFeature` is a pure virtual class that defines a single custom docking constraint used by `OE Dock`.

`OEFeature` objects are created by the `OE CustomConstraint` class (see `OE CustomConstraint::AddFeature`).

To satisfy a docking constraint a pose must either have:

1. At least one atom that matches one of the SMARTS patterns returned by `OEFeature::GetSMARTS` and falls within one of the spheres returned by `OEFeature::GetSpheres`.

- At least one heavy atom that falls within one of the spheres returned by `OEFeature::GetSpheres`, if the `OEFeature` has no SMARTS patterns.

operator=

```
OEFeature& operator=(const OEFeature&)
```

Assignment operator.

Destructor

```
virtual ~OEFeature() {}
```

Destructor.

GetSpheres

```
virtual OESystem::OEIterBase<const OESphereBase>* GetSpheres() = 0
virtual OESystem::OEIterBase<OESphereBase>* GetSpheres() = 0
```

Returns all spheres associated with this custom constraint feature.

AddSphere

```
virtual OESphereBase* AddSphere() = 0
virtual OESphereBase* AddSphere(const OESphereBase& sph) = 0
```

Adds an `OESphereBase` 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

```
virtual bool DeleteSphere(const OESphereBase* sph) = 0
```

Deletes a `OESphereBase` held by this class. Note that after this call *sph* will no longer point to a valid `OESphereBase` object.

NumSpheres

```
virtual unsigned int NumSpheres() const = 0
```

Returns the number of spheres held by this object.

ClearSpheres

```
virtual bool ClearSpheres() = 0
```

Returns this object to its default constructed state. Deleting all spheres and SMARTS patterns it currently holds.

GetSMARTS

```
virtual OESystem::OEIterBase<const std::string>* GetSMARTS() const = 0  
virtual OESystem::OEIterBase<std::string>* GetSMARTS() = 0
```

Returns an iterator over all SMARTS patterns associated with this `OESystem`.

AddSMARTS

```
virtual bool AddSMARTS(std::string smarts) = 0
```

Adds a SMARTS pattern to this `OESystem`.

DeleteSMARTS

```
virtual bool DeleteSMARTS(std::string smarts) = 0
```

Deletes the specified SMARTS pattern from this `OESystem`.

NumSMARTS

```
virtual unsigned int NumSMARTS() const = 0
```

Returns the number of SMARTS patterns held by this `OESystem`.

ClearSMARTS

```
virtual bool ClearSMARTS() = 0
```

Deletes all SMARTS patterns associated with this `OESystem`.

SetFeatureName

```
virtual void SetFeatureName(std::string) = 0
```

Sets the name of this feature.

GetFeatureName

```
virtual std::string GetFeatureName() const = 0
```

Returns the name of the feature.

SetEnabled

```
virtual void SetEnabled(bool enabled) = 0
```

Sets whether this feature is enabled or not. If the feature is disabled the constraint will be ignored during the docking process.

GetEnabled

```
virtual bool GetEnabled() const = 0
```

Returns whether this feature is enabled. If this function returns false this constraint will be ignored during the docking process.

CreateCopy

```
virtual OEFeature* CreateCopy() const = 0
```

Creates a copy of this `OEFeature`. Note that the memory returned by this function is **not** owned by this class, and is the responsibility of the calling function.

4.1.6 OEHybrid

```
class OEHybrid : public OEDock
```

Note: This `OEHybrid` is identical to `OEDock` except that the default `OEDockMethod` is `OEDockMethod::Hybrid`.

`OEHybrid` is used to dock and score multiconformer molecules in an active site using the *Hybrid Method*.

Use the following procedure to dock and score molecules with this class.

1. Construct the object.
2. Initialize the active site by passing a receptor (see *Receptors* chapter) to the `OEHybrid::Initialize` method.
3. Dock molecules using the `OEHybrid::DockMultiConformerMolecule` method.
4. Score docked molecules with any of the following methods
 - `OEHybrid::ScoreLigand`
 - `OEHybrid::ScoreAtom`
 - `OEHybrid::ScoreLigandComponent`
 - `OEHybrid::ScoreAtomComponent`

- `OEHybrid::AnnotatePose`

Constructor

```
OEHybrid(unsigned int scoring = OEDockMethod::Hybrid,
         unsigned int resolution = OESearchResolution::Default)
```

Constructs the `OEHybrid` object to use a given *scoring* method and search *resolution*.

Table 4.2: Parameters

| | |
|------------|--|
| scoring | The scoring method, from the <code>OEDockMethod</code> namespace, this object will use. |
| resolution | The search resolution, from the <code>OESearchResolution</code> namespace, this object will use. |

IsInitialized

```
bool IsInitialized() const
```

Returns true if this object has been successfully initialized and is ready to dock molecules.

Initialize

```
bool Initialize(const OEChem::OEMolBase& receptor)
```

Sets up this object to dock molecules into a receptor.

This function returns true if setup was successful and false otherwise.

receptor A receptor describing the active site of the target protein (see *Receptors* chapter). This object does not depend upon receptor once this call is completed (*i.e.* receptor can be destroyed after calling this method).

Note: This method can take several minutes to complete with larger active sites.

DockMultiConformerMolecule

Both overloads of this method dock *inputMol* into the receptor passed to the `OEHybrid::Initialize` method.

The return value of this method describes the result of the docking, with a value from the `OEDockingReturnCode` namespace. A results of `OEDockingReturnCode::Success` indicates docking was successful.

```
unsigned int DockMultiConformerMolecule(OEChem::OEMolBase& dockedMol,
                                         const OEChem::OEMCMolBase& inputMol)
```

This overload of `OEHybrid::DockMultiConformerMolecule` returns the top scoring pose.

dockedMol Top scoring docked pose of *inputMol*.

inputMol A multiconformer representation of a molecule to dock.

The score of the docked pose can be obtained by calling the `OEMolBase::GetEnergy` method of *dockedMol*.

```
unsigned int DockMultiConformerMolecule(OEChem::OEMolBase& dockedMol,
                                         const OEChem::OEMolBase& inputMol,
                                         unsigned int numPoses = 1)
```

This overload of `OEHybrid::DockMultiConformerMolecule` can return alternate docked poses, in addition to the top scoring pose.

the top scoring poses.

dockedMol Docked poses of *inputMol*. Poses are stored as conformers of the `OEChem::OEMolBase` and are sorted by score.

inputMol A multiconformer representation of a molecule to dock.

numPoses Maximum number of top scoring docked poses to return in *dockedMol*. Typically this will be the number of poses returned, however, in highly restricted sites fewer than *numPoses* may be returned. The value of *numPoses* must be greater than zero.

The score of the docked poses can be obtained by calling the `OEMolBase::GetEnergy` method on the conformers of *dockedMol*.

GetHighScoresAreBetter

```
bool GetHighScoresAreBetter() const
```

Returns true if higher scores indicate a better result.

Returns false if lower scores indicate a better result.

GetName

```
std::string GetName() const ;
```

Returns the name of the scoring functions docked poses are scored with.

GetComponentNames

```
OESystem::OEIterBase<const std::string>* GetComponentNames() const
```

ScoreLigand

```
float ScoreLigand(const OEChem::OEMolBase& pose)
```

Rescores a pose within the active site.

pose Structure of a pose within the active site

If an error occurs this function will return `FLT_MAX` if `OEHybrid::GetHighScoresAreBetter` returns false or `-FLT_MAX` otherwise.

ScoreAtom

```
float ScoreAtom(const OEChem::OEAtomBase& atom,
               const OEChem::OEMolBase& pose)
```

Returns the score of an atom of a given pose within the active site.

atom Atom of *pose* to score.

pose Structure of a pose within the active site

If an error occurs this function will return FLT_MAX if `OEHybrid::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

ScoreLigandComponent

```
float ScoreLigandComponent(const OEChem::OEMolBase& pose,
                          std::string compName)
```

Returns the given components contribution to the total score.

pose Structure of a pose within the active site

compName Name of the score component to report. Name must be one returned by `OEHybrid::GetComponentNames`.

If an error occurs this function will return FLT_MAX if `OEHybrid::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

ScoreAtomComponent

```
float ScoreAtomComponent(const OEChem::OEAtomBase& atom,
                        const OEChem::OEMolBase& pose,
                        std::string compName)
```

Returns the given components contribution to the score of a given atom score.

atom Atom of *pose* to score.

pose Structure of a pose within the active site

compName Name of the score component to report. Name must be one returned by `OEHybrid::GetComponentNames`.

If an error occurs this function will return FLT_MAX if `OEHybrid::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

AnnotatePose

Adds **VIDA** scoring annotation to the *pose* or *poses* passed.

The annotated poses must be written out in either *oeb* or *oeb.gz* format and are only viewable in **VIDA**.

```
bool AnnotatePose(OEChem::OEMolBase& pose)
```

This overload of `OEHybrid::AnnotatePose` annotates a single pose.

pose Structure of a pose within the active site

```
bool AnnotatePose(OEChem::OEMCMolBase& poses)
```

This overload of `OEHybrid::AnnotatePose` annotates all poses of a given ligand.

poses An `OEMCMolBase` the conformers of which are poses within the active site.

CacheScoringSetup

This function caches the current scoring setup of the `OEHybrid` object onto a receptor object. When another `OEHybrid` object is initialized with this receptor the it will read in the cached score data rather than recalculating it from scratch, thus improving the startup time of the `OEHybrid` object.

```
bool CacheScoringSetup(OEChem::OEMolBase& receptor, bool clearOldData = true)
```

receptor A receptor object. This must be the same receptor object the `OEHybrid` object was initialized with or an exact copy.

clearOldData Flag to clear cached data from a prior call of `CacheScoringSetup` with this receptor. The cached data can be quite sizeable (hundreds of megabytes), so leaving this flag at the default value of true is recommended.

Note: The cached score data on a receptor will be saved when the receptor is written to a file.

4.1.7 OEProteinConstraint

```
class OEProteinConstraint
```

This class holds information about a single receptor protein constraint.

GetAtom

```
OEChem::OEAtomBase* GetAtom() const
```

Hold the protein atom of the receptor the constraint is associated with.

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.

4.1.8 OEScore

```
class OEScore
```

`OEScore` is a class for scoring and optimizing poses within the active site. Use of this class is as follows:

1. Choose a scoring function at construction time by passing one of the following constants to the constructor
 - `OEScoreType::Chemgauss3`
 - `OEScoreType::Chemscore`
 - `OEScoreType::PLP`
 - `OEScoreType::Shapegauss`
2. Initialize the active site by passing either a receptor (see *Receptors* chapter) or a protein and box to the `OEScore::Initialize` method.
3. Score or optimize the molecules using:
 - `OEScore::ScoreLigand`

- `OEScore::ScoreAtom`
- `OEScore::ScoreLigandComponent`
- `OEScore::ScoreAtomComponent`
- `OEScore::SystematicSolidBodyOptimize`
- `OEScore::AnnotatePose`

Note: The following free function can take `OEScore` objects.

| | |
|-------------------------------|--|
| <code>OESetEnergyScore</code> | <code>OESetEnergyScoreComponent</code> |
| <code>OESetSDScore</code> | <code>OESetSDScoreComponent</code> |
| <code>OESetScore</code> | <code>OESetScoreComponent</code> |

Constructor

```
OEScore(const unsigned int scoring = OEScoreType::Default)
```

Constructs `OEScore` to use the specified *scoring* function from the `OEScoreType` namespace.

IsInitialized

```
bool IsInitialized() const
```

Returns true if this object has been successfully initialized and is ready to score poses.

Initialize

Sets up this object to score poses within an active site. Initialization can be done with either a receptor or a protein and box enclosing the active site.

Both overloads return true if setup was successful.

```
bool Initialize(const OEChem::OEMolBase& receptor)
```

This overload of `OEScore::Initialize` takes a receptor.

receptor A receptor describing the active site of the target protein (see *Receptors* chapter). This object does not depend upon receptor once this call is completed (*i.e.* receptor can be destroyed after calling this method).

```
bool Initialize(const OEChem::OEMolBase& protein,
               const OEBoxBase& box)
```

This overload of `OEScore::Initialize` takes a protein and box.

protein The structure of a target protein. This object does not depend upon *receptor* once this call is completed (*i.e.* *protein* can be destroyed after calling this method).

box An `OEBoxBase` enclosing active site.

GetName

```
std::string GetName() const
```

Returns the name of the scoring function.

GetComponentNames

```
OESystem::OEIterBase<const std::string>* GetComponentNames() const
```

Returns the name of the scoring function components.

GetHighScoresAreBetter

```
bool GetHighScoresAreBetter() const
```

Returns true if higher scores indicate a better result.

Returns false if lower scores indicate a better result.

ScoreLigand

```
float ScoreLigand(const OEChem::OEMolBase& pose)
```

Scores a pose within the active site.

pose Structure of a pose within the active site

If an error occurs this function will return FLT_MAX if `OEChem::OEScore::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

ScoreAtom

```
float ScoreAtom(const OEChem::OEAtomBase& atom,
                const OEChem::OEMolBase& pose)
```

Returns the score of an atom of a given pose within the active site.

atom Atom of *pose* to score.

pose Structure of a pose within the active site

If an error occurs this function will return FLT_MAX if `OEChem::OEScore::GetHighScoresAreBetter` returns false or -FLT_MAX otherwise.

ScoreLigandComponent

```
float ScoreLigandComponent(const OEChem::OEMolBase& pose,
                           std::string compName)
```

Returns the given component's contribution to the total score.

pose Structure of a pose within the active site

compName Name of the score component to report. Name must be one returned by `OEScore::GetComponentNames`.

If an error occurs this function will return `FLT_MAX` if `OEScore::GetHighScoresAreBetter` returns false or `-FLT_MAX` otherwise.

ScoreAtomComponent

```
float ScoreAtomComponent(const OEChem::OEAtomBase& atom,
                        const OEChem::OEMolBase& pose,
                        std::string compName)
```

Returns the given components contribution to the score of a given atom score.

atom Atom of *pose* to score.

pose Structure of a pose within the active site

compName Name of the score component to report. Name must be one returned by `OEScore::GetComponentNames`.

If an error occurs this function will return `FLT_MAX` if `OEScore::GetHighScoresAreBetter` returns false or `-FLT_MAX` otherwise.

SystematicSolidBodyOptimize

Performs a solid body optimization on the *pose* or *poses* passed. Three rotational and three translational degrees of freedom are explored, the pose (and protein) are held rigid. A plus and minus rotational step is taken for each rotational and translational degree of freedom (for a total of 3^6 or 729 positions tested for each pose).

Both overloads return true if no error occurred.

```
bool SystematicSolidBodyOptimize(OEChem::OEMCMolBase& poses,
                                unsigned int searchResolution
                                = OESearchResolution::Default)
```

This overload of `OEScore::SystematicSolidBodyOptimize` optimizes all poses of an `OEMCMolBase`.

poses An `OEMCMolBase` with conformers that are poses within the active site to be optimized.

searchResolution Search resolution of the optimization (see `OESearchResolution` constant namespace).

The optimized poses are returned in *poses*. This poses retain thier original ordering and are not ordered by the new optimized score. The simplest method to reorder these new optimized poses is to call the function `OESetEnergyScore` followed by `OESortConfsByEnergy`.

```
bool SystematicSolidBodyOptimize(OEChem::OEMolBase& pose,
                                unsigned int searchResolution
                                = OESearchResolution::Default)
```

This overload of `OEScore::SystematicSolidBodyOptimize` optimizes a single pose.

pose A pose within the active site to be optimized.

searchResolution Search resolution of the optimization (see `OESearchResolution` constant namespace).

AnnotatePose

Adds **VIDA** scoring annotation to the *pose* or *poses* passed.

The annotated poses must be written out in either *oeb* or *oeb.gz* format and are only viewable in **VIDA**.

bool AnnotatePose(OEChem::OEMolBase& pose)

This overload of `OEScore::AnnotatePose` annotates a single pose.

pose Structure of a pose within the active site

bool AnnotatePose(OEChem::OEMCMolBase& poses)

This overload of `OEScore::AnnotatePose` annotates all poses of a given ligand.

poses An `OEMCMolBase` the conformers of which are poses within the active site.

CacheScoringSetup

This function caches the current scoring setup of the `OEScore` object onto a receptor object. When another `OEScore` object is initialized with this receptor the it will read in the cached score data rather than recalculating it from scratch, thus improving the startup time of the `OEScore` object.

bool CacheScoringSetup(OEChem::OEMolBase& receptor, **bool** clearOldData = **true**)

receptor A receptor object. This must be the same receptor object the `OEScore` object was initialized with or an exact copy.

clearOldData Flag to clear cached data from a prior call of `CacheScoringSetup` with this receptor. The cached data can be quite sizeable (hundreds of megabytes), so leaving this flag at the default value of `true` is recommended.

Note: The cached score data on a receptor will be saved when the receptor is written to a file.

4.1.9 OESphereBase

`OESphereBase` class represents a sphere.

The following free functions take `OESphereBase`.

| | |
|-----------------------------|---------------------------|
| <code>OESphereVolume</code> | <code>OESphereArea</code> |
| <code>OEInSphere</code> | |

Destructor

virtual ~OESphereBase() {}

Destructor.

operator=

```
OESphereBase& operator=(const OESphereBase& rhs)
```

Assignment operator.

GetX

```
virtual float GetX() const = 0
```

Returns the X coordinate of the center of the sphere

GetY

```
virtual float GetY() const = 0
```

Returns the Y coordinate of the center of the sphere

GetZ

```
virtual float GetZ() const = 0
```

Return the Z coordinate of the center of the sphere

GetRad

```
virtual float GetRad() const = 0
```

Returns the radius of the sphere

SetRad

```
virtual bool SetRad(float rad) = 0
```

Sets the radius of the sphere to *rad*. Will fail and return false if *rad* < 0.

SetCenter

```
virtual bool SetCenter(float x,  
                      float y,  
                      float z) = 0
```

Sets the center of the sphere.

4.2 OEDocking Constants

4.2.1 OEDockingReturnCode

A constant from this namespace is returned by `OEDock::DockMultiConformerMolecule` to indicate the outcome of the docking.

Success

Docking was successful.

NotInitialized

Docking failed because the `OEDock` object has not been initialized (see `OEDock::Initialize`)

EmptyLigand

Docking failed because the ligand contained no atoms.

EmptyProtein

The protein passed to the `OEDock::Initialize` method contained no atoms.

NoValidPoses

No ligand poses could fit within the active site. Increasing the size of the receptor's outer contour volume (see *Negative Image* section) and re-initializing the `OEDock` object with the new receptor may allow the ligand to dock.

NoConstraintMatch

The supplied ligand cannot match the docking constraints.

TypeError

One or more of the atoms on the ligand could not be typed.

GridSetupError

One of the scoring grids could not be setup. This generally indicates that the protein structure is broken.

CoordError

The geometry of the chemical interactions on the ligand could not be determined. This generally indicates that the ligand is broken in some way (e.g. valence errors).

4.2.2 OEDockMethod

Constants in this namespace are used by the constructor of `OEDock` and `OEHybrid` (see `OEDock::Constructor` and `OEHybrid::Constructor` respectively) to specify the scoring function used during the exhaustive search and optimization and final scoring.

Shapegauss

Exhaustive Search `Shapegauss`

Optimization and Scoring `Shapegauss`

PLP

Exhaustive Search `PLP`

Optimization and Scoring `PLP`

Chemgauss3

Exhaustive Search `Chemgauss3`

Optimization and Scoring `Chemgauss3`

Chemgauss4

Exhaustive Search `Chemgauss3`

Optimization and Scoring `Chemgauss4`

Chemgauss

Same as `OEDockMethod::Chemgauss4`

Chemscore

Exhaustive Search `Chemgauss3`

Optimization and Scoring `Chemscore`

Hybrid1

Exhaustive Search *Chemical Gaussian Overlay*

Optimization and Scoring *Chemgauss3*

Hybrid2

Exhaustive Search *Chemical Gaussian Overlay*

Optimization and Scoring *Chemgauss4*

Hybrid

Same as `OEDockMethod::Hybrid2`

Default

Same as `OEDockMethod::Chemgauss4`

INVALID

Return value used to indicate an error

4.2.3 OESearchResolution

Constants in this namespace are used by the constructor of `OEDock` (see `OEDock::Constructor`) to specify the search resolution used during the exhaustive search and optimization as well as the number of poses passed from the exhaustive search to the optimization step. It is also used by the `OEScore::SystematicSolidBodyOptimize` method to specify the resolution of the optimization (the exhaustive search values are ignored in this case).

Rotational stepsize is the furthest distance any atom will move in a single rotational step.

All distances are measured in angstroms.

High

Exhaustive Search Resolution

Translational 1.0

Rotational 1.0

Optimization Resolution

Translational 0.5

Rotational 0.5

Number exhaustive search poses optimized during docking

1000

Standard

Exhaustive Search Resolution

Translational 1.0

Rotational 1.5

Optimization Resolution

Translational 0.5

Rotational 0.75

Number exhaustive search poses optimized during docking

100

Low

Exhaustive Search Resolution

Translational 1.5

Rotational 2.0

Optimization Resolution

Translational 0.75

Rotational 1.0

Number exhaustive search poses optimized during docking

100

Default

Same as `OESearchResolution::Standard`

INVALID

Used as a return type to indicate an error

4.2.4 OEProteinConstraintType

Constants in this namespace are used by the `OEProteinConstraint` 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.

4.2.5 OENegativeImageType

These constants are used by the `OEMakeReceptor` function to specify the type of negative image it should make.

LargeShape

Create a negative image designed for larger sites. This type of negative image tends to prefer solvent exposed areas over tight cavities.

StandardShape

Standard negative image appropriate for most sites

SmallShape

Create a negative image designed for smaller sites. This type of negative image tends to prefer tight cavities over more solvent exposed areas.

Default

Same as `OENegativeImageType::StandardShape`

INVALID

Return value used to indicate an error

4.2.6 OEScoreType

Constants in this namespace are used by the constructor of `OEScore` (see `OEScore::Constructor`) to specify the scoring function `OEScore` will use.

Shapegauss

Shapegauss.

PLP

PLP.

Chemgauss3

Chemgauss3.

Chemgauss4

Chemgauss4.

Chemgauss

Same as `OEScoreType::Chemgauss4`.

Chemscore

Chemscore.

Default

Same as `OEScoreType::Chemgauss4`.

INVALID

Used as a return type to indicate an error.

4.3 OEDocking Functions

4.3.1 OEBoxBase functions

OEBoxExtend

```
bool OEBoxExtend(OEBoxBase& box, float x_ext, float y_ext, 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(OEBoxBase& box, float ext)
```

Extends every face of *box* by *ext*. Thus every dimension of the box will increase by 2 times *ext*.

OEBoxTranslate

```
bool OEBoxTranslate(OEBoxBase& box, float x_trans, float y_trans, float z_trans)
```

Translates *box* a distance specified by (*x_trans*, *y_trans*, *z_trans*)

OEBoxVolume

```
float OEBoxVolume(const OEBoxBase& box)
```

Return the volume of the box

OEBoxArea

```
float OEBoxArea(const OEBoxBase& box)
```

Return the area of the box.

OEInBox

```
bool OEInBox(const OEBoxBase& box, float x, float y, 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*.

OEBoxXMid

```
float OEBoxXMid(const OEBoxBase& box)
```

Returns the X midpoint of the box.

OEBoxYMid

```
float OEBoxYMid(const OEBoxBase& box)
```

Returns the Y midpoint of the box.

OEBoxZMid

```
float OEBoxZMid(const OEBoxBase& box)
```

Returns the Z midpoint of the box.

OEBoxXDim

```
float OEBoxXDim(const OEBoxBase& box)
```

Returns the X dimension of the box.

OEBoxYDim

```
float OEBoxYDim(const OEBoxBase& box)
```

Returns the Y dimension of the box.

OEBoxZDim

```
float OEBoxZDim(const OEBoxBase& box)
```

Returns the Z dimension of the box.

OESetupBox

```
bool OESetupBox(OEBoxBase& box, float* xyz, unsigned int N, 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.

```
bool OESetupBox(OEBoxBase& box, const OEChem::OEMolBase& mol, float addbox = 0.0f)
```

Sets up *box* to be the minimum size box that encloses *mol* and then extends each box face by *addbox*.

box Box to setup

mol A molecule the box will be setup around

addbox Optional parameter to extend each box face by a specified amount relative to the minimum box enclosing the *mol*.

Returns true if setup was successful.

```
bool OESetupBox(OEBoxBase& box, const OEChem::OEMCMolBase& mol, float addbox = 0.0f)
```

Sets up *box* to be the minimum size box that encloses all conformers of *mol* and then extends each box face by *addbox*.

box Box to setup

mol A multiconformer molecule the box will be setup around

addbox Optional parameter to extend each box face by a specified amount relative to the minimum box enclosing the *mol*.

Returns true if setup was successful.

```
bool OESetupBoxCenterAndExtents(OEBoxBase& box, float* center, 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.

OEMakeBoxMolecule

```
bool OEMakeBoxMolecule(OEChem::OEMolBase& mol, const OEBoxBase& box)
```

Creates a molecule (*mol*) out of *box*. *mol* will have 8 carbon atoms, one at each corner of the box and a single bond between appropriate atoms.

Note that this molecule is in no way chemically valid. The purpose of this method is to create a molecule representing *box* that can be viewed in a molecular visualizer.

4.3.2 OEDockMethod namespace functions

OEDockMethodGetName

```
string OEDockMethodGetName(unsigned int method)
```

Returns the name of the scoring function. *method* is a constant from the `OEDockMethod` namespace.

OEDockMethodGetValue

```
unsigned int OEDockMethodGetValue(std::string name)
```

Returns the value of a constant from the `OEDockMethod` namespace associated with the scoring function *name*, or `OEDockMethod::INVALID` if the name is not recognized.

```
unsigned int OEDockMethodGetValue(const OESystem::OEInterface& itf, std::string flagName)
```

Used in conjunction with the `OEDockMethodConfigure` function. Returns a constant from the `OEDockMethod` namespace associated with the dock method the user specified on the command line, or `OEDockMethod::Default` if the user did not specify a value. *itf* and *flagName* should be the same as those passed to the `OEDockMethodConfigure` function and `OEParseCommandLine` should have been called on *itf*.

OEDockMethodConfigure

```
bool OEDockMethodConfigure(OESystem::OEInterface& itf, std::string flagName)
```

This function is used in conjunction with the `OEDockMethodGetValue`. It adds a command line flag to the *itf* to specify a dock method to use. Once the command line has been parsed (`OEParseCommandLine`) the *itf* and *flagName* are passed to `OEDockMethodGetValue` that will return the constant from the `OEDockMethod` namespace associated with the dock method the user selected (or the default scoring function if the user didn't specify one).

Note: The parameter added to *itf* will be of type string, name *flagName*, and have legal values associated with the dock method.

4.3.3 OESearchResolution namespace functions

OESearchResolutionGetName

```
string OESearchResolutionGetName(unsigned int resolution)
```

Returns the name of the scoring function. *resolution* is a constant from the `OESearchResolution` namespace.

OESearchResolutionGetValue

```
unsigned int OESearchResolutionGetValue(std::string name)
```

Returns the value of a constant from the `OESearchResolution` namespace associated with the scoring function *name*, or `OESearchResolution::INVALID` if the name is not recognized.

```
unsigned int OESearchResolutionGetValue(const OESystem::OEInterface& itf,
                                        std::string flagName)
```

Used in conjunction with the `OESearchResolutionConfigure` function. Returns a constant from the `OESearchResolution` namespace associated with the dock resolution the user specified on the command line, or `OESearchResolution::Default` if the user did not specify a value. *itf* and *flagName* should be the same as those passed to the `OESearchResolutionConfigure` function and should have been called on *itf*.

OESearchResolutionConfigure

```
bool OESearchResolutionConfigure(OESystem::OEInterface& itf, std::string flagName)
```

This function is used in conjunction with the `OESearchResolutionGetValue`. It adds a command line flag to the *itf* to specify a dock resolution to use. Once the command line has been parsed (`OEParseCommandLine`) the *itf* and *flagName* are passed to `OESearchResolutionGetValue` that will return the constant from the `OESearchResolution` namespace associated with the dock resolution the user selected (or the default scoring function if the user didn't specify one).

Note: The parameter added to *itf* will be of type string, name *flagName*, and have legal values associated with the dock resolutions.

4.3.4 OENegativeImageType namespace functions

OENegativeImageTypeGetName

```
string OENegativeImageTypeGetName(unsigned int method)
```

Returns the name of the scoring function. *method* is a constant from the `OENegativeImageType` namespace.

OENegativeImageTypeGetValue

```
unsigned int OENegativeImageTypeGetValue(std::string name)
```

Returns the value of a constant from the `OENegativeImageType` namespace associated with the scoring function *name*, or `OENegativeImageType::INVALID` if the name is not recognized.

```
unsigned int OENegativeImageTypeGetValue(const OESystem::OEInterface& itf,  
std::string flagName)
```

Used in conjunction with the `OENegativeImageTypeConfigure` function. Returns a constant from the `OENegativeImageType` namespace associated with the negative image type the user specified on the command line, or `OENegativeImageType::Default` if the user did not specify a value. *itf* and *flagName* should be the same as those passed to the `OENegativeImageTypeConfigure` function and `OEParseCommandLine` should have been called on *itf*.

OENegativeImageTypeConfigure

```
bool OENegativeImageTypeConfigure(OESystem::OEInterface& itf, std::string flagName)
```

This function is used in conjunction with the `OENegativeImageTypeGetValue`. It adds a command line flag to the *itf* to specify a negative image type to use. Once the command line has been parsed (`OEParseCommandLine`) the *itf* and *flagName* are passed to `OENegativeImageTypeGetValue` that will return the constant from the `OENegativeImageType` namespace associated with the negative image type the user selected (or the default scoring function if the user didn't specify one).

Note: The parameter added to *itf* will be of type string, name *flagName*, and have legal values associated with the negative image type.

4.3.5 OEScoreType namespace functions

OEScoreTypeGetName

```
string OEScoreTypeGetName(unsigned int scoring)
```

Returns the name of the scoring function. *scoring* is a constant from this namespace.

OEScoreTypeGetValue

```
unsigned int OEScoreTypeGetValue(std::string name)
```

Returns the value of a constant from this namespace associated with the scoring function *name*, or `OEScoreType::INVALID` if the name is not recognized.

This function is defined within the `OEScoreType` namespace.

```
unsigned int OEScoreTypeGetValue(const OESystem::OEInterface& itf, std::string flagName)
```

Used in conjunction with the `OEScoreTypeConfigure` function. Returns a constant from this namespace associated with the scoring function the user specified on the command line, or `OEScoreType::Default` if the user did not specify a value. *itf* and *flagName* should be the same as those passed to the `OEScoreTypeConfigure` function and `OEParseCommandLine` should have been called on *itf*.

OEScoreTypeConfigure

```
bool OEScoreTypeConfigure(OESystem::OEInterface& itf, std::string flagName)
```

This function is used in conjunction with the `OEScoreTypeGetValue`. It adds a command line flag to the *itf* to specify a scoring function to use. Once the command line has been parsed (`OEParseCommandLine`) the *itf* and *flagName* are passed to `OEScoreTypeGetValue` that will return the constant from this namespace associated with the scoring function the user selected (or the default scoring function if the user didn't specify one).

Note: The parameter added to *itf* will be of type string, name *flagName*, and have legal values associated with the scoring functions.

4.3.6 Receptor API

OEIsReceptor

```
bool OEIsReceptor(const OEChem::OEMolBase& receptor)
```

Returns true if *rec* is a receptor.

OEReceptorClear

```
bool OEReceptorClear(OEChem::OEMolBase& receptor)
```

Clears all receptor information from *receptor*, except the structure of the protein.

4.3.7 Receptor Negative Image API

These functions deal with the receptor's negative image, used by `OEDock` during the docking process (see *Negative Image* section).

OEReceptorHasNegativeImageGrid

```
bool OEReceptorHasNegativeImageGrid(const OEChem::OEMolBase& receptor)
```

Returns true if the *receptor* has a negative image grid.

OEReceptorGetNegativeImageGrid

```
OESystem::OEGrid<float> OEReceptorGetNegativeImageGrid(const OEChem::OEMolBase& receptor)
```

Returns a copy of the *receptor*'s negative image grid.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorHasOuterContourLevel

```
bool OEReceptorHasOuterContourLevel(const OEChem::OEMolBase& receptor)
```

Returns true if the *receptor* has an outer contour level set.

Note: This function will return true even if the outer contour is disabled (i.e. even if the level is negative).

OEReceptorGetOuterContourLevel

```
float OEReceptorGetOuterContourLevel(const OEChem::OEMolBase& receptor)
```

Returns the outer contour *level* of the *receptor*.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorSetOuterContourLevel

```
bool OEReceptorSetOuterContourLevel(OEChem::OEMolBase& receptor, float level)
```

Sets the outer contour *level* on the *receptor*.

Returns true if successful.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorHasInnerContourLevel

```
bool OEReceptorHasInnerContourLevel(const OEChem::OEMolBase& receptor)
```

Returns true if the *receptor* has an inner contour level set.

Note: This function will return true even if the inner contour is disabled (*i.e.* even if the level is negative).

OEReceptorGetInnerContourLevel

```
float OEReceptorGetInnerContourLevel(const OEChem::OEMolBase& receptor, float& level)
```

Returns the inner contour *level* of the *receptor*.

It is only valid to call this function if *receptor* is a valid receptor (*i.e.*, `OEIsReceptor` returns true).

OEReceptorSetInnerContourLevel

```
bool OEReceptorSetInnerContourLevel(OEChem::OEMolBase& receptor, float level)
```

Sets the inner contour *level* on the *receptor*.

Returns true if successful.

It is only valid to call this function if *receptor* is a valid receptor (*i.e.*, `OEIsReceptor` returns true).

4.3.8 Receptor Bound Ligand API

OEReceptorHasBoundLigand

```
bool OEReceptorHasBoundLigand(const OEChem::OEMolBase& receptor)
```

Returns true if *receptor* is a valid receptor and has a bound ligand.

OEReceptorGetBoundLigand

```
OEChem::OEMolBase& OEReceptorGetBoundLigand(const OEChem::OEMolBase& receptor)
```

Returns a copy of the *receptor's* bound ligand.

If the receptor does not have a bound ligand then an empty molecule will be returned

It is only valid to call this function if *receptor* is a valid receptor (*i.e.*, `OEIsReceptor` returns true).

OEReceptorSetBoundLigand

```
bool OEReceptorSetBoundLigand(OEChem::OEMolBase& receptor,
                               const OEChem::OEMolBase& boundLigand)
```

Sets the bound ligand on *receptor* to be a copy of *boundLigand*.

Returns true if successful.

It is only valid to call this function if *receptor* is a valid receptor (*i.e.*, `OEIsReceptor` returns true).

OEReceptorClearBoundLigand

```
bool OEReceptorClearBoundLigand(OEChem::OEMolBase& receptor)
```

Clears any bound ligand from *receptor*.

Returns true if successful.

4.3.9 Receptor Extra Molecules API

OEReceptorHasExtraMols

```
bool OEReceptorHasExtraMols(const OEChem::OEMolBase& receptor)
```

Returns true if the *receptor* is a valid receptor and has one or more extra molecules.

OEReceptorGetExtraMols

```
OEIterBase<const OEMolBase>* OEReceptorGetExtraMols(const OEChem::OEMolBase& receptor)
```

Returns an iterator over the extra molecules held by *receptor*.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorAddExtraMol

```
bool OEReceptorAddExtraMol(OEChem::OEMolBase& receptor,  
                             const OEChem::OEMolBase& extraMol)
```

Adds a copy of *extraMol* to *receptor*'s extra molecules.

Returns true if successful.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorClearExtraMols

```
bool OEReceptorClearExtraMols(OEChem::OEMolBase& receptor)
```

Clears all extra molecules from *receptor*.

Returns true if successful.

4.3.10 Receptor Protein Constraint API

OEReceptorHasProteinConstraints

```
bool OEReceptorHasProteinConstraints(const OEChem::OEMolBase& receptor,  
                                       bool enabledOnly = true)
```

Returns true if *receptor* is a valid receptor and has one or more protein constraints. If *enabledOnly* the constraint(s) must also be enabled.

OEReceptorGetProteinConstraints

```
OEIterBase<OEProteinConstraint>*
OEReceptorGetProteinConstraint (const OEChem::OEMolBase& receptor,
                               bool enabledOnly = true)
```

Returns an iterator over the *receptor's* protein constraints. If *enabledOnly* is true then only enabled constraints will be returned.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorSetProteinConstraint

```
bool OEReceptorSetProteinConstraint (OEChem::OEMolBase& receptor,
                                     OEProteinConstraint& proteinConstraint)
```

Sets a protein constraint. If the *receptor* already has a constraint on `OEProteinConstraint::GetAtom` the constraint will be replaced, otherwise a new one will be created.

Returns true if successful.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorClearProteinConstraint

```
bool OEReceptorClearProteinConstraints (OEChem::OEMolBase& receptor,
                                         const OEProteinConstraint& constraint)
```

Clears a given protein constraint from *receptor*.

OEReceptorClearProteinConstraints

```
bool OEReceptorClearProteinConstraints (OEChem::OEMolBase& receptor)
```

Clears all protein constraints from *receptor*.

4.3.11 Receptor Custom Constraint API

OEReceptorHasCustomConstraints

```
bool OEReceptorHasCustomConstraints (const OEChem::OEMolBase& receptor,
                                     bool enabledOnly = true)
```

Returns true if *receptor* has custom constraints. If *enabledOnly* is true then at least one of the constraints must be enabled.

OEReceptorGetCustomConstraints

```
bool OEReceptorGetCustomConstraints(const OEChem::OEMolBase& receptor,
                                     OECustomConstraintsBase& customConstraints)
```

Returns a copy of the *receptor*'s custom constraints

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorSetCustomConstraints

```
bool OEReceptorSetCustomConstraints(OEChem::OEMolBase& receptor,
                                     const OECustomConstraintsBase& customConstraints)
```

Sets the *receptor*'s custom constraints (copied from *customConstraints*).

Returns true if successful.

It is only valid to call this function if *receptor* is a valid receptor (i.e., `OEIsReceptor` returns true).

OEReceptorClearCustomConstraints

```
bool OEReceptorClearCustomConstraints(OEChem::OEMolBase& receptor)
```

Clears all custom constraints on a receptor.

Returns true if successful.

4.3.12 Receptor Score Setup Cache API

OEReceptorHasCachedScore

```
bool OEReceptorHasCachedScore(const OEChem::OEMolBase& rec)
```

Returns true if *rec* has a score setup cache for any `OEScore` or `OEDock` object.

OEReceptorAddCachedScore

```
bool OEReceptorAddCachedScore(OEChem::OEMolBase& rec, OEScore& score)
```

Equivalent to calling `OEScore::CacheScoringSetup` with *clearOldData* false.

```
bool OEReceptorAddCachedScore(OEChem::OEMolBase& rec, OEDock& dock)
```

Equivalent to calling `OEDock::CacheScoringSetup` with *clearOldData* false.

OEReceptorSetCachedScore

```
bool OEReceptorSetCachedScore(OEChem::OEMolBase& rec, OEScore& score)
```

Equivalent to calling `OEScore::CacheScoringSetup` with `clearOldData` true.

```
bool OEReceptorSetCachedScore(OEChem::OEMolBase& rec, OEDock& dock)
```

Equivalent to calling `OEDock::CacheScoringSetup` with `clearOldData` true.

OEReceptorClearCachedScore

```
bool OEReceptorClearCachedScore(const OEChem::OEMolBase& rec)
```

Clears all score setup caches, from `OEScore` or `OEDock` objects, from the receptor.

4.3.13 Create Receptor Functions

OEMakeReceptor

```
bool OEMakeReceptor(OEChem::OEMolBase& receptor,
                   const OEChem::OEMolBase& proteinStructure,
                   const OEBoxBase& box,
                   bool stripWater = true,
                   const unsigned int negImgType = OENegativeImageType::Default)
```

Creates a *receptor* object using a *box* and the protein structure.

receptor: The created receptor.

proteinStructure: Structure of a target protein.

box: An `OEBoxBase` enclosing the active site on the *proteinStructure*. The box should enclose all possible docked ligand positions (heavy atoms of the docked pose must fit within the box). It is not necessary to enclose the protein residues the docked ligand may interact with.

stripWater: If this flag is true water molecules present in *proteinStructure* will be stripped from the protein structure of the *receptor* and added to the extra molecules information of the *receptor*.

negImgType: Specifies the type of negative image to create (see `OENegativeImageType` namespace).

This function returns true if successful.

```
bool OEMakeReceptor(OEChem::OEMolBase& receptor,
                   const OEChem::OEMolBase& proteinStructure,
                   const OEChem::OEMolBase& boundLigand,
                   bool stripWater = true,
                   unsigned int negImgType = OENegativeImageType::Default)
```

Creates a *receptor* object using a *boundLigand* and the protein structure.

receptor: The created receptor.

proteinStructure: Structure of a target protein.

boundLigand: The structure of a ligand bound to the active site.

stripWater: If this flag is true water molecules present in *proteinStructure* will be stripped from the protein structure of the *receptor* and added to the extra molecules information of the *receptor*.

negImgType: Specifies the type of negative image to create (see [OENegativeImageType](#) namespace).

This function returns true if successful.

```
bool OEMakeReceptor(OEChem::OEMolBase& receptor,
                   const OEChem::OEMolBase& proteinStructure,
                   const OEChem::OEAtomBase& hint,
                   bool stripWater = true,
                   unsigned int negImgType = OENegativeImageType::Default)
```

Creates a *receptor* object using the protein structure and the identity of one atom of the protein structure near the active site.

receptor: The created receptor.

proteinStructure: Structure of a target protein.

hint: An atom of *proteinStructure* that is near the binding site.

stripWater: If this flag is true water molecules present in *proteinStructure* will be stripped from the protein structure of the *receptor* and added to the extra molecules information of the *receptor*.

negImgType: Specifies the type of negative image to create (see [OENegativeImageType](#) namespace).

This function returns true if successful.

```
bool OEMakeReceptor(OEChem::OEMolBase& receptor,
                   const OEChem::OEMolBase& proteinStructure,
                   float hintX,
                   float hintY,
                   float hintZ,
                   bool stripWater = true,
                   unsigned int negImgType = OENegativeImageType::Default)
```

Creates a *receptor* object using the protein structure and a coordinate near the active site.

receptor: The created receptor.

proteinStructure: Structure of a target protein.

hintX hintY hintZ: Coordinate near the binding site of *proteinStructure*.

stripWater: If this flag is true water molecules present in *proteinStructure* will be stripped from the protein structure of the *receptor* and added to the extra molecules information of the *receptor*.

negImgType: Specifies the type of negative image to create (see [OENegativeImageType](#) namespace).

4.3.14 Receptor file I/O

OEReadReceptorFile

```
bool OEReadReceptorFile(OEChem::OEMolBase& receptor, std::string filename)
```

Reads a receptor from file *filename* into *receptor*.

Returns true if successful.

Note: All receptor files must be in either .oeb or .oeb.gz format.

OEWriteReceptorFile

```
bool OEWriteReceptorFile(const OEChem::OEMolBase& receptor, std::string filename)
bool OEWriteReceptorFile(OEChem::OEMolBase& receptor, std::string filename)
```

Writes a *receptor* to file *filename*

Returns true if successful.

Note: All receptor files must be in either .oeb or .oeb.gz format.

4.3.15 Score assignment functions

These functions are convenience functions for attaching scores to molecules in either SD data, float generic data or the energy field of the molecule. These functions call either the **ScoreLigand** or **ScoreLigandComponent** method of the **OEDock** or **OEScore** object passed to them in order to obtain the scores.

OESetEnergyScore

```
bool OESetEnergyScore(OEChem::OEMolBase& pose,
                      OEDock& dock)
```

Sets the energy of *pose* (OEMolBase::GetEnergy) to the score of the pose calculated with the **OEDock::ScoreLigand** method of *dock*.

```
bool OESetEnergyScore(OEChem::OEMolBase& pose,
                      OEScore& score)
```

Sets the energy of *pose* (OEMolBase::GetEnergy) to the score of the pose calculated with the **OEScore::ScoreLigand** method of *score*.

```
bool OESetEnergyScore(OEChem::OEMCMolBase& poses,
                      OEDock& dock,
                      bool sortPoses = false)
```

Sets the energy of each pose of *poses* (OEMolBase::GetEnergy) to the score of the pose calculated with the **OEDock::ScoreLigand** method of *dock*.

If *sortPoses* is true the poses of will also be sorted by score.

```
bool OESetEnergyScore(OEChem::OEMCMolBase& poses,  
                      OEScore& score,  
                      bool sortPoses = false)
```

Sets the energy of each pose of *poses* (OEMolBase::GetEnergy) to the score of the pose calculated with the OEScore::ScoreLigand method of *score*.

If *sortPoses* is true the poses will also be sorted by score.

OESetSDScore

```
bool OESetSDScore(OEChem::OEMolBase& pose,  
                  OEDock& dock,  
                  std::string sdtag)
```

Assigns the score of *pose*, using the OEDock::ScoreLigand method of *dock*, to SD data of *pose* with the given *sdtag*.

```
bool OESetSDScore(OEChem::OEMolBase& pose,  
                  OEScore& score,  
                  std::string sdtag)
```

Assigns the score of *pose*, using the OEScore::ScoreLigand method of *score*, to SD data of *pose* with the given *sdtag*.

```
bool OESetSDScore(OEChem::OEMCMolBase& poses,  
                  OEDock& dock,  
                  std::string sdtag,  
                  bool sortPoses = false)
```

Assigns the score of each pose of *poses*, using the OEDock::ScoreLigand method of *dock*, to SD data of the pose with the given *sdtag*.

If *sortPoses* is true the poses will also be sorted by score.

```
bool OESetSDScore(OEChem::OEMCMolBase& poses,  
                  OEScore& score,  
                  std::string sdtag,  
                  bool sortPoses = false)
```

Assigns the score of each pose of *poses*, using the OEScore::ScoreLigand method of *score*, to SD data of the pose with the given *sdtag*.

If *sortPoses* is true the poses will also be sorted by score.

OESetScore

```
bool OESetScore(OEChem::OEMolBase& pose,  
                OEDock& dock,  
                std::string tag)
```

Assigns the score of *pose*, using the OEDock::ScoreLigand method of *dock*, to float generic data of *pose* with the given *tag*.

```
bool OESetScore (OEChem::OEMolBase&,
                OEScore& score,
                std::string tag)
```

Assigns the score of *pose*, using the `OEScore::ScoreLigand` method of *score*, to float generic data of *pose* with the given *tag*.

```
bool OESetScore (OEChem::OEMCMolBase& poses,
                OEDock& dock,
                std::string tag,
                bool sortPoses = false)
```

Assigns the score of each pose of *poses*, using the `OEDock::ScoreLigand` method of *dock*, to float generic data of the pose with the given *tag*.

If *sortPoses* is true the poses will also be sorted by score.

```
bool OESetScore (OEChem::OEMCMolBase& poses,
                OEScore& score,
                std::string tag,
                bool sortPoses = false)
```

Assigns the score of each pose of *poses*, using the `OEScore::ScoreLigand` method of *score*, to float generic data of the pose with the given *tag*.

If *sortPoses* is true the poses will also be sorted by score.

OESetEnergyScoreComponent

```
bool OESetEnergyScoreComponent (OEChem::OEMolBase& pose,
                                OEDock& dock,
                                std::string component)
```

Sets the energy of *pose* (`OEMolBase::GetEnergy`) to the *component* of the score calculated with *dock*'s `OEDock::ScoreLigandComponent` method.

```
bool OESetEnergyScoreComponent (OEChem::OEMolBase& pose,
                                OEScore& score,
                                std::string component)
```

Sets the energy of *pose* (`OEMolBase::GetEnergy`) to the *component* of the score calculated with *score*'s `OEScore::ScoreLigandComponent` method.

```
bool OESetEnergyScoreComponent (OEChem::OEMCMolBase& pose,
                                OEDock& dock,
                                std::string component,
                                bool sortPoses = false)
```

Sets the energy of each pose of *poses* (`OEMolBase::GetEnergy`) to the *component* of the score calculate with *dock*'s `OEDock::ScoreLigandComponent` method.

If *sortPoses* is true the poses will also be sorted by score.

```
bool OESetEnergyScoreComponent (OEChem::OEMCMolBase& pose,
                                OEScore& score,
                                std::string component,
                                bool sortPoses = false)
```

Sets the energy of each pose of *poses* (`OEChem::OEMolBase::GetEnergy`) to the *component* of the score calculated with *score*'s `OEScore::ScoreLigandComponent` method.

If *sortPoses* is true the poses will also be sorted by score.

OESetSDScoreComponent

```
bool OESetSDScoreComponent (OEChem::OEMolBase& pose,
                             OEDock& dock,
                             std::string component,
                             std::string sdtag)
```

Assigns the *component* of *pose*'s score, calculated with *dock*'s `OEDock::ScoreLigandComponent` method, to *pose*'s SD data (with tag *sdtag*).

```
bool OESetSDScoreComponent (OEChem::OEMolBase& pose,
                             OEScore& score,
                             std::string component,
                             std::string sdtag)
```

Assigns the *component* of *pose*'s score, calculated with *score*'s `OEScore::ScoreLigandComponent` method, to *pose*'s SD data (with tag *sdtag*).

```
bool OESetSDScoreComponent (OEChem::OEMCMolBase& poses,
                             OEDock& dock,
                             std::string component,
                             std::string sdtag,
                             bool sortPoses = false)
```

Assigns the *component* of the score of each pose of *poses*, calculated with *dock*'s `OEDock::ScoreLigandComponent` method, to SD data (with tag *sdtag*) on the pose.

If *sortPoses* is true the poses will also be sorted by score.

```
bool OESetSDScoreComponent (OEChem::OEMCMolBase& poses,
                             OEScoreBase& score,
                             std::string component,
                             std::string sdtag,
                             bool sortPoses = false)
```

Assigns the *component* of the score of each pose of *poses*, calculated with *score*'s `OEScore::ScoreLigandComponent` method, to SD data (with tag *sdtag*) on the pose.

If *sortPoses* is true the poses will also be sorted by score.

OESetScoreComponent

```
bool OESetScoreComponent (OEChem::OEMolBase& pose,
                          OEDock& dock,
                          std::string component,
                          std::string tag)
```

Assigns the *component* of *pose*'s score, calculated with *dock*'s `OEDock::ScoreLigandComponent` method, to *pose*'s float generic data (with tag *tag*) on *pose*.

```
bool OESetScoreComponent (OEChem::OEMolBase& pose,
                          OEScore& score,
                          std::string component,
                          std::string tag)
```

Assigns the *component* of *pose*'s score, calculated with *scores*'s `OEScore::ScoreLigandComponent` method, to *pose*'s float generic data (with tag *tag*) on *pose*.

```
bool OESetScoreComponent (OEChem::OEMCMolBase& pose,
                          OEDock& dock,
                          std::string component,
                          std::string tag,
                          bool sortPoses = false)
```

Assigns the *component* of the score of each pose of *poses*, calculated with *dock*'s `OEDock::ScoreLigandComponent` method, to float generic data (with tag *tag*) on the pose.

If *sortPoses* is true the poses will also be sorted by score.

```
bool OESetScoreComponent (OEChem::OEMCMolBase& poses,
                          OEScore& score,
                          std::string component,
                          std::string tag,
                          bool sortPoses = false)
```

Assigns the *component* of the score of each pose of *poses*, calculated with *score*'s `OEScore::ScoreLigandComponent` method, to float generic data (with tag *tag*) on the pose.

If *sortPoses* is true the poses will also be sorted by score.

4.3.16 OESphereBase functions

OESphereVolume

```
float OESphereVolume(const OESphereBase& sph)
```

OESphereArea

```
float OESphereArea(const OESphereBase& sph)
```

OESphere

```
bool OESphere(const OESphereBase& sph, float x, float y, float z)
```

RELEASE NOTES

5.1 DockingTK 1.1.1

5.1.1 Bug fixes

- Fixed bug where SD data on the input molecules was not copied to the docked output molecule.
- Fixed memory leak when scoring a molecule that does not fit within the receptor site.
- Fixed an invalid memory access bug that occurred in rare cases when docking molecules.

5.2 DockingTK 1.1.0

- The behavior of `OEDockMethod::Hybrid` from version 1.0 has been changed. `OEDockMethod::Hybrid` is now an alias for the most up to date docking method, currently `OEDockMethod::Hybrid2`. `OEDockMethod::Hybrid1` now provides the same functionality that `OEDockMethod::Hybrid` did in version 1.0 of *DockingTK*.

5.2.1 New Features

- *DockingTK* is now thread safe. For details on thread safety see the “Thread Safety” section of the OEChem toolkit.
- *Chemgauss4* scoring function, available both as a docking method (`OEDockMethod::Chemgauss4`) and scoring method (`OEScoreType::Chemgauss4`).
- Hybrid 2 docking method (`OEDockMethod::Hybrid2`), that uses *Chemgauss4* for the structure based portion of the docking.
- `OEHybrid` class for doing hybrid docking added for convenience. `OEHybrid` is identical to `OEDock`, except that the default method is `OEDockMethod::Hybrid` rather than `OEDockMethod::Chemgauss`.
- Scoring function setup information from `OEDock`, `OEHybrid` and `OEScore` objects can now be saved on receptor files, decreasing the setup time for future runs.
- Fixed a bug where OEB handlers were not setup correctly sometimes due to link ordering problems.

5.2.2 Bug fixes

- Fixed bug generating negative image grids, that caused the negative image to extend further from the protein and penetrate less deeply into pockets than was intended.
- Fixed a bug in the local optimization (done during docking or with a call to `OEScore::SystematicSolidBodyOptimize`) that caused the local rotations to be less evenly spaced than they could be. This bug also affected the optimization done by the `OEDock` class.

5.3 DockingTK 1.0.0

- Docking features
 - Exhaustive Search docking followed by pose optimization
 - Hybrid docking (uses the structure a known bound active to guide docking)
 - Docking constraints (e.g., require specific hydrogen bonding interactions)
- Scoring features
 - Score optimization (systematic solid body optimization)
 - Break down of score by atom and/or scoring function component
 - Score annotating (stores score breakdown on molecule for visualization in VIDA)
- Implementation of four scoring functions
 - Chemgauss3
 - Chemscore
 - PLP
 - Shapegauss

BIBLIOGRAPHY

BIBLIOGRAPHY

- [McGann-2003] Mark McGann, Harold R Almond, Anthony Nicholls, J. Andrew Grant and Frank K. Brown, **Gaussian Docking Functions**, *BioPolymers*, Vol. 68, pp. 76-90, **2003**
- [Verkivker-2000] Gennady M. Verkivker, Djamal Bouzida, Daniel K. Gehlaar, Paul A. Rejto, Sandra Arthurs, Anthony B. Colson, Stephan T. Freer, Veda Larson, Brock A. Luty, Tami Marrone and Peter W. Rose, **Deciphering common failures in molecular docking of ligand-protein complexes**, *Journal of Computer-Aided Molecular Design*, Vol. 14, pp. 731-751, **2000**
- [Eldridge-1997] Matthew D. Eldridge, Christopher W. Murray, Timothy R. Auton, Gaia V. Paolini and Roger P. Mee., **Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes**, *Journal of Computer-Aided Molecular Design*, Vol. 11, pp. 425-445, **1997**

INDEX

Symbols

- in
 - command line option, 13, 21
- method
 - command line option, 21
- optimize
 - command line option, 13
- out
 - command line option, 13, 21
- receptor
 - command line option, 13, 21
- resolution
 - command line option, 21
- score
 - command line option, 13

A

- AddFeature
 - OEDocking::OECustomConstraint, 32
- AddSMARTS
 - OEDocking::OEFeature, 38
- AddSphere
 - OEDocking::OEFeature, 37
- AnnotatePose
 - OEDocking::OEDock, 36
 - OEDocking::OEHybrid, 42
 - OEDocking::OEScore, 48

C

- CacheScoringSetup
 - OEDocking::OEDock, 36
 - OEDocking::OEHybrid, 43
 - OEDocking::OEScore, 48
- Clear
 - OEDocking::OECustomConstraint, 32
- ClearSMARTS
 - OEDocking::OEFeature, 38
- ClearSpheres
 - OEDocking::OEFeature, 38
- command line option
 - in, 13, 21

- method, 21
- optimize, 13
- out, 13, 21
- receptor, 13, 21
- resolution, 21
- score, 13

Constructor

- OEDocking::OEDock, 33
- OEDocking::OEHybrid, 40
- OEDocking::OEScore, 45

Constructors

- OEDocking::OEBox, 29
- OEDocking::OECustomConstraint, 31

CreateCopy

- OEDocking::OEFeature, 39

D

- DeleteFeature
 - OEDocking::OECustomConstraint, 32
- DeleteSMARTS
 - OEDocking::OEFeature, 38
- DeleteSphere
 - OEDocking::OEFeature, 37
- Destructor
 - OEDocking::OEBoxBase, 27
 - OEDocking::OEFeature, 37
 - OEDocking::OESphereBase, 48
- DockMolecules.cpp
 - Example Code, 21
- DockMultiConformerMolecule
 - OEDocking::OEDock, 34
 - OEDocking::OEHybrid, 40

E

- Example Code
 - DockMolecules.cpp, 21
 - RescorePoses.cpp, 14

G

- GetAtom
 - OEDocking::OEProteinConstraint, 43

GetComponentNames
 OEDocking::OEDock, 35
 OEDocking::OEHybrid, 41
 OEDocking::OEScore, 46

GetEnabled
 OEDocking::OEFeature, 39
 OEDocking::OEProteinConstraint, 43

GetFeatureName
 OEDocking::OEFeature, 39

GetFeatures
 OEDocking::OECustomConstraint, 31

GetHighScoresAreBetter
 OEDocking::OEDock, 34
 OEDocking::OEHybrid, 41
 OEDocking::OEScore, 46

GetName
 OEDocking::OEDock, 34
 OEDocking::OEHybrid, 41
 OEDocking::OEProteinConstraint, 44
 OEDocking::OEScore, 46

GetRad
 OEDocking::OESphereBase, 49

GetSMARTS
 OEDocking::OEFeature, 38

GetSpheres
 OEDocking::OEFeature, 37

GetType
 OEDocking::OEProteinConstraint, 43

GetX
 OEDocking::OESphereBase, 49

GetXMax
 OEDocking::OEBox, 30
 OEDocking::OEBoxBase, 27

GetXMin
 OEDocking::OEBox, 30
 OEDocking::OEBoxBase, 28

GetY
 OEDocking::OESphereBase, 49

GetYMax
 OEDocking::OEBox, 30
 OEDocking::OEBoxBase, 28

GetYMin
 OEDocking::OEBox, 30
 OEDocking::OEBoxBase, 28

GetZ
 OEDocking::OESphereBase, 49

GetZMax
 OEDocking::OEBox, 30
 OEDocking::OEBoxBase, 28

GetZMin
 OEDocking::OEBox, 30
 OEDocking::OEBoxBase, 28

I

Initialize
 OEDocking::OEDock, 33
 OEDocking::OEHybrid, 40
 OEDocking::OEScore, 45

IsInitialized
 OEDocking::OEDock, 33
 OEDocking::OEHybrid, 40
 OEDocking::OEScore, 45

N

NumFeatures
 OEDocking::OECustomConstraint, 32

NumSMARTS
 OEDocking::OEFeature, 38

NumSpheres
 OEDocking::OEFeature, 37

O

OEDocking::OEBox, 28
 Constructors, 29
 GetXMax, 30
 GetXMin, 30
 GetYMax, 30
 GetYMin, 30
 GetZMax, 30
 GetZMin, 30
 operator=, 30
 Setup, 31

OEDocking::OEBoxArea, 56

OEDocking::OEBoxBase, 27
 Destructor, 27
 GetXMax, 27
 GetXMin, 28
 GetYMax, 28
 GetYMin, 28
 GetZMax, 28
 GetZMin, 28
 operator=, 27
 Setup, 28

OEDocking::OEBoxExtend, 56

OEDocking::OEBoxTranslate, 56

OEDocking::OEBoxVolume, 56

OEDocking::OEBoxXDim, 57

OEDocking::OEBoxXMid, 56

OEDocking::OEBoxYDim, 57

OEDocking::OEBoxYMid, 57

OEDocking::OEBoxZDim, 57

OEDocking::OEBoxZMid, 57

OEDocking::OECustomConstraint, 31
 AddFeature, 32
 Clear, 32
 Constructors, 31

DeleteFeature, 32
 GetFeatures, 31
 NumFeatures, 32
 operator=, 31
 OEDocking::OEDock, 32
 AnnotatePose, 36
 CacheScoringSetup, 36
 Constructor, 33
 DockMultiConformerMolecule, 34
 GetComponentNames, 35
 GetHighScoresAreBetter, 34
 GetName, 34
 Initialize, 33
 IsInitialized, 33
 ScoreAtom, 35
 ScoreAtomComponent, 35
 ScoreLigand, 35
 ScoreLigandComponent, 35
 OEDocking::OEDockingReturnCode, 50
 OEDocking::OEDockingReturnCode::CoordError, 51
 OEDocking::OEDockingReturnCode::EmptyLigand, 50
 OEDocking::OEDockingReturnCode::EmptyProtein, 50
 OEDocking::OEDockingReturnCode::GridSetupError, 50
 OEDocking::OEDockingReturnCode::NoConstraintMatch, 50
 OEDocking::OEDockingReturnCode::NotInitialized, 50
 OEDocking::OEDockingReturnCode::NoValidPoses, 50
 OEDocking::OEDockingReturnCode::Success, 50
 OEDocking::OEDockingReturnCode::TypingError, 50
 OEDocking::OEDockMethod, 51
 OEDocking::OEDockMethod::Chemgauss, 51
 OEDocking::OEDockMethod::Chemgauss3, 51
 OEDocking::OEDockMethod::Chemgauss4, 51
 OEDocking::OEDockMethod::Chemscore, 51
 OEDocking::OEDockMethod::Default, 52
 OEDocking::OEDockMethod::Hybrid, 52
 OEDocking::OEDockMethod::Hybrid1, 52
 OEDocking::OEDockMethod::Hybrid2, 52
 OEDocking::OEDockMethod::INVALID, 52
 OEDocking::OEDockMethod::PLP, 51
 OEDocking::OEDockMethod::Shapegauss, 51
 OEDocking::OEDockMethodConfigure, 59
 OEDocking::OEDockMethodGetName, 58
 OEDocking::OEDockMethodGetValue, 59
 OEDocking::OEFeature, 36
 AddSMARTS, 38
 AddSphere, 37
 ClearSMARTS, 38
 ClearSpheres, 38
 CreateCopy, 39
 DeleteSMARTS, 38
 DeleteSphere, 37
 Destructor, 37
 GetEnabled, 39
 GetFeatureName, 39
 GetSMARTS, 38
 GetSpheres, 37
 NumSMARTS, 38
 NumSpheres, 37
 operator=, 37
 SetEnabled, 39
 SetFeatureName, 38
 OEDocking::OEHybrid, 39
 AnnotatePose, 42
 CacheScoringSetup, 43
 Constructor, 40
 DockMultiConformerMolecule, 40
 GetComponentNames, 41
 GetHighScoresAreBetter, 41
 GetName, 41
 Initialize, 40
 IsInitialized, 40
 ScoreAtom, 42
 ScoreAtomComponent, 42
 ScoreLigand, 41
 ScoreLigandComponent, 42
 OEDocking::OEInBox, 56
 OEDocking::OEInSphere, 74
 OEDocking::OEIsReceptor, 61
 OEDocking::OEMakeBoxMolecule, 58
 OEDocking::OEMakeReceptor, 67
 OEDocking::OENegativeImageType, 54
 OEDocking::OENegativeImageType::Default, 54
 OEDocking::OENegativeImageType::INVALID, 55
 OEDocking::OENegativeImageType::LargeShape, 54
 OEDocking::OENegativeImageType::SmallShape, 54
 OEDocking::OENegativeImageType::StandardShape, 54
 OEDocking::OENegativeImageTypeConfigure, 60
 OEDocking::OENegativeImageTypeGetName, 60
 OEDocking::OENegativeImageTypeGetValue, 60
 OEDocking::OEProteinConstraint, 43
 GetAtom, 43
 GetEnabled, 43
 GetName, 44
 GetType, 43
 SetAtom, 44
 SetEnabled, 44
 SetName, 44
 SetType, 44
 OEDocking::OEProteinConstraintType, 53
 OEDocking::OEProteinConstraintType::Acceptor, 54
 OEDocking::OEProteinConstraintType::Chelator, 53
 OEDocking::OEProteinConstraintType::Contact, 54
 OEDocking::OEProteinConstraintType::Donor, 54
 OEDocking::OEProteinConstraintType::Lipophilic, 54
 OEDocking::OEProteinConstraintType::Unknown, 54
 OEDocking::OEReadReceptorFile, 69

OEDocking::OEReceptorAddCachedScore, 66
 OEDocking::OEReceptorAddExtraMol, 64
 OEDocking::OEReceptorClear, 61
 OEDocking::OEReceptorClearBoundLigand, 64
 OEDocking::OEReceptorClearCachedScore, 67
 OEDocking::OEReceptorClearCustomConstraints, 66
 OEDocking::OEReceptorClearExtraMols, 64
 OEDocking::OEReceptorClearProteinConstraint, 65
 OEDocking::OEReceptorClearProteinConstraints, 65
 OEDocking::OEReceptorGetBoundLigand, 63
 OEDocking::OEReceptorGetCustomConstraints, 66
 OEDocking::OEReceptorGetExtraMols, 64
 OEDocking::OEReceptorGetInnerContourLevel, 63
 OEDocking::OEReceptorGetNegativeImageGrid, 62
 OEDocking::OEReceptorGetOuterContourLevel, 62
 OEDocking::OEReceptorGetProteinConstraints, 65
 OEDocking::OEReceptorHasBoundLigand, 63
 OEDocking::OEReceptorHasCachedScore, 66
 OEDocking::OEReceptorHasCustomConstraints, 65
 OEDocking::OEReceptorHasExtraMols, 64
 OEDocking::OEReceptorHasInnerContourLevel, 62
 OEDocking::OEReceptorHasNegativeImageGrid, 62
 OEDocking::OEReceptorHasOuterContourLevel, 62
 OEDocking::OEReceptorHasProteinConstraints, 64
 OEDocking::OEReceptorSetBoundLigand, 63
 OEDocking::OEReceptorSetCachedScore, 67
 OEDocking::OEReceptorSetCustomConstraints, 66
 OEDocking::OEReceptorSetInnerContourLevel, 63
 OEDocking::OEReceptorSetOuterContourLevel, 62
 OEDocking::OEReceptorSetProteinConstraint, 65
 OEDocking::OEScore, 44
 AnnotatePose, 48
 CacheScoringSetup, 48
 Constructor, 45
 GetComponentNames, 46
 GetHighScoresAreBetter, 46
 GetName, 46
 Initialize, 45
 IsInitialized, 45
 ScoreAtom, 46
 ScoreAtomComponent, 47
 ScoreLigand, 46
 ScoreLigandComponent, 46
 SystematicSolidBodyOptimize, 47
 OEDocking::OEScoreType, 55
 OEDocking::OEScoreType::Chemgauss, 55
 OEDocking::OEScoreType::Chemgauss3, 55
 OEDocking::OEScoreType::Chemgauss4, 55
 OEDocking::OEScoreType::Chemgauss4, 55
 OEDocking::OEScoreType::Chemscore, 55
 OEDocking::OEScoreType::Default, 55
 OEDocking::OEScoreType::INVALID, 55
 OEDocking::OEScoreType::PLP, 55
 OEDocking::OEScoreType::Shapegauss, 55
 OEDocking::OEScoreTypeConfigure, 61
 OEDocking::OEScoreTypeGetName, 61
 OEDocking::OEScoreTypeGetValue, 61
 OEDocking::OESearchResolution, 52
 OEDocking::OESearchResolution::Default, 53
 OEDocking::OESearchResolution::High, 52
 OEDocking::OESearchResolution::INVALID, 53
 OEDocking::OESearchResolution::Low, 53
 OEDocking::OESearchResolution::Standard, 53
 OEDocking::OESearchResolutionConfigure, 60
 OEDocking::OESearchResolutionGetName, 59
 OEDocking::OESearchResolutionGetValue, 59
 OEDocking::OESetEnergyScore, 69
 OEDocking::OESetEnergyScoreComponent, 71
 OEDocking::OESetScore, 70
 OEDocking::OESetScoreComponent, 72
 OEDocking::OESetSDScore, 70
 OEDocking::OESetSDScoreComponent, 72
 OEDocking::OESetupBox, 57
 OEDocking::OESphereArea, 73
 OEDocking::OESphereBase, 48
 Destructor, 48
 GetRad, 49
 GetX, 49
 GetY, 49
 GetZ, 49
 operator=, 49
 SetCenter, 49
 SetRad, 49
 OEDocking::OESphereVolume, 73
 OEDocking::OEWriteReceptorFile, 69
 operator=
 OEDocking::OEBox, 30
 OEDocking::OEBoxBase, 27
 OEDocking::OECustomConstraint, 31
 OEDocking::OEFeature, 37
 OEDocking::OESphereBase, 49

R

RescorePoses.cpp
 Example Code, 14

S

ScoreAtom
 OEDocking::OEDock, 35
 OEDocking::OEHybrid, 42
 OEDocking::OEScore, 46
 ScoreAtomComponent
 OEDocking::OEDock, 35
 OEDocking::OEHybrid, 42
 OEDocking::OEScore, 47
 ScoreLigand
 OEDocking::OEDock, 35
 OEDocking::OEHybrid, 41
 OEDocking::OEScore, 46

ScoreLigandComponent
 OEDocking::OEDock, 35
 OEDocking::OEHybrid, 42
 OEDocking::OEScore, 46

SetAtom
 OEDocking::OEProteinConstraint, 44

SetCenter
 OEDocking::OESphereBase, 49

SetEnabled
 OEDocking::OEFeature, 39
 OEDocking::OEProteinConstraint, 44

SetFeatureName
 OEDocking::OEFeature, 38

SetName
 OEDocking::OEProteinConstraint, 44

SetRad
 OEDocking::OESphereBase, 49

SetType
 OEDocking::OEProteinConstraint, 44

Setup
 OEDocking::OEBox, 31
 OEDocking::OEBoxBase, 28

SystematicSolidBodyOptimize
 OEDocking::OEScore, 47