silicos

Open Babel 2.3 C++ API Quick reference guide

Namespace

```
using namespace OpenBabel;
```

Reading and writing

```
#include <openbabel/mol.h>
#include <openbabel/obconversion.h>

int main(int argc, char **argv)
{
   OBMol mol;

   // Read from standard input and write to
   // standard output
   OBConversion conv(&std::cin, &std::cout);

   // Input format is sd-file, output format
   // is canonical smiles
   if (conv.SetInAndOutFormats("sdf", "can"))
   {
      if (conv.Read(&mol))
      {
         // Print number of atoms
         std::cerr << "Molecule has: " <<
         mol.NumAtoms() << " atoms.\n";
      }
      conv->Write(&mol);
      mol.Clear();
   }
   return 0;
}
```

Conversion

```
#include <openbabel/obconversion.h>
#include <openbabel/mol.h>

// Create molecule from SMILES string
std::string SmilesString("c1ccccc1");
OBMol mol;
stringstream ss(SmilesString)
OBConversion conv(&ss);
if (conv.SetInFormat("smi") && conv.Read(&mol))
{ /* ... */ }

// Conversion without manipulation
OBConversion conv(&is, &os);
if (conv.SetInAndOutFormats("SMI", "MOL"))
{
   // Option "h" adds explicit hydrogens
   conv.AddOption("h", OBConversion::GENOPTIONS);
   conv.Convert();
}
```

```
// Automatic format perception
std::ifstream ifs(filename);
OBConversion conv;
OBFormat* inFormat = conv.FormatFromExt(filename);
OBFormat* outFormat = conv.SetFormat("SDF");
if (inFormat && outFormat)
{
   conv.SetInAndOutFormats(inFormat, outFormat);
}
```

Looping over atoms and bonds

```
#include <openbabel/obiter.h>
#include <openbabel/mol.h>

using namespace OpenBabel;

OBMol mol;
OBAtom* atom;
OBAtom* nbrAtom;
OBBond* bond;

// Looping over all atoms
double exactMass(0.0);
FOR_ATOMS_OF_MOL(atom, mol)
{
   exactMass += atom->GetExactMass();
}

// Looping over all bonds
unsigned int totalBondOrder(0);
FOR_BONDS_OF_MOL(bond, mol)
{
   totalBondOrder += bond->GetBondOrder();
}
// Looping over all neighbor atoms of given atom
unsigned int nAtoms(0);
FOR_NBORS_OF_ATOM(nbrAtom, atom) ++nAtoms;
```

Element table

```
#include <openbabel/data.h>

OBElementTable etab;

(char*)  etab.GetSymbol(6);
(int)    etab.GetAtomicNum("C");
(double) etab.GetVdwRad(7);
(double) etab.GetCovalentRad(8);
(double) etab.GetMass(1);
```

Molecules

```
#include <openbabel/mol.h>
#include <openbabel/generic.h>

OBMol mol;

// Number of atoms and bonds
(unsigned int) mol.NumAtoms();    // All atoms
(unsigned int) mol.NumHvyAtoms(); // Non-H atoms
(unsigned int) mol.NumBonds();    // All bonds

// Molecular weight with implicit hydrogens
(double) mol.GetMolWt(true);
```

```
// Molecular formula
(std::string) mol.GetFormula();

// Conformations
(int) mol.NumConformers();

// Smallest set of smallest rings
(std::vector<OBRing*>) mol.GetSSSR();

// Hydrogen manipulations
(bool) mol.DeleteHydrogens();   // All H
(bool) mol.AddPolarHydrogens(); // Polar H
(bool) mol.AddHydrogens();      // All H

// Generic data
std::vector<OBGenericData*>::iterator k;
std::vector<OBGenericData*> vdata = mol.GetData();
for (k = vdata.begin(); k != vdata.end(); ++k)
if ((*k)->GetDataType() == OBGenericDataType::PairData)
{
   std::cout << ">  <" << (*k)->GetAttribute();
   std::cout << ">" << std::endl;
   std::cout << ((OBPairData*)(*k))->GetValue();
   std::cout << std::endl;
}

// Remove all but the largest fragments
(bool) mol.StripSalts(0);

// Clear molecule for re-use
(bool) mol.Clear();
```

Atoms

```
#include <openbabel/atom.h>

OBAtom atom;

// Properties
(int) atom.GetFormalCharge();
(unsigned int) atom.GetAtomicNum();
(double) atom.GetAtomicMass();

// Explicit and maximum expected connections
(unsigned int) atom.GetValence();
(unsigned int) atom.GetImplicitValence();

// Non-hydrogen connections
(unsigned int) atom.GetHvyValence();

// Implicit and explicit hydrogens
(unsigned int) atom.ImplicitHydrogenCount();
(unsigned int) atom.ExplicitHydrogenCount();

// Is atom in ring of size 6?
(bool) atom.IsInRing() && atom.IsInRingSize(6);

// Size of smallest ring that contains the atom
(unsigned int) atom.MemberOfRingSize();

// Number of rings that contain the atom
(unsigned int) atom.MemberOfRingCount();
```

Bonds

```
#include <openbabel/bond.h>
#include <openbabel/atom.h>
```

```
OBBond bond;

// Properties
(unsigned int) bond.GetBondOrder();
(bool) bond.IsPrimaryAmide(); // and 2 and 3...
(bool) bond.IsSingle(); // and 2 and 3...
(bool) bond.IsRotor();

// Flanking atoms
(OBAtom*) bond.GetBeginAtom();
(OBAtom*) bond.GetEndAtom();
OBAtom* atom; (OBAtom*) bond.GetNbrAtom(atom);

// Is bond in ring?
(bool) bond.IsInRing();
```

Substructure search

```
#include <openbabel/parsmart.h>
#include <openbabel/mol.h>
#include <openbabel/atom.h>

OBMol mol;
OBAtom* atom;
/* ... */

// Create a SMARTS pattern of a phenyl ring
OBSmartsPattern sp;
sp.Init("c1ccccc1");

// Properties of the substructure
if (sp.IsValid())
{
   std::cout << sp.NumAtoms();
   std::cout << sp.NumBonds();
   std::cout << sp.GetSmarts();
}

// Matching
(bool) sp.Match(mol, true);  // Single matching
(bool) sp.Match(mol, false); // Complete matching

// Substructure mapping
std::vector<std::vector<int> > mapListA;
mapListA = sp.GetMapList(); // Non-unique matches
std::vector<std::vector<int> > mapListU;
mapListU = sp.GetUMapList(); // Unique matches
for (int m(0); m < mapListU.size(); ++m)
{
   std::cout << "Unique match " << m << std::endl;
   for (int a(0); a < mapListU[m].size(); ++a)
   {
      atom = mol.GetAtom(mapListU[m][a]);
      std::cout << atom->GetAtomicNum() << std::endl;
   }
}
```

Spectrophores™

```
#include <openbabel/obspectrophore.h>
#include <openbabel/mol.h>

OBMol mol;
/* ... */

// Create a Spectrophore object
OBSpectrophore spec;

// Set calculation parameters
spec.SetResolution(3.0);
spec.SetAccuracy(AngStepSize20);
spec.SetStereo(NoStereoSpecificProbes);
spec.SetNormalization(NormalizationTowardsZeroMean);

// Calculate and print
std::vector<double> sphore;
sphore = spec.GetSpectrophore(&mol);
for (int i(0); i < sphore.size(); ++i)
{
   std::cout << sphore[i] << "\t";
}
std::cout << std::endl;
```

Stereochemistry

```
#include <openbabel/mol.h>
#include <openbabel/obconversion.h>
#include <openbabel/stereo/tetrahedral.h>

OBMol mol;
OBConversion conv;
conv.SetInFormat("smi");
conv.ReadString(&mol, "C[C@H](Cl)Br");

// Stereofacade object
OBStereoFacade facade(&mol);
(unsigned int) facade.NumTetrahedralStereo();
(unsigned int) facade.NumCisTransStereo();
(unsigned int) facade.SquarePlanarStereo();

// Loop over all atoms to check if stereocenter
FOR_ATOMS_OF_MOL(atom, mol)
{
   std::cout << atom->GetId() << ": ";
   if (facade.HasTetrahedralStereo(atom->GetId()))
      std::cout << ": stereo";
   else
      std::cout << ": no stereo";
   std::cout << std::endl;
}
```

Rings

```
#include <openbabel/mol.h>
#include <openbabel/ring.h>
#include <openbabel/math/vector3.h>

OBMol mol;
/* ... */

// Get the smallest-set-of-smallest-rings
std::vector<OBRing*> rings = mol.GetSSSR();
vector3 center;
vector3 normal_up;
vector3 normal_down;
for (int i(0); i < rings.size(); ++i)
{
   (bool) rings[i].IsAromatic();
   (int)  rings[i].Size();
   (bool) rings[i].findCenterAndNormal(center,
                                       normal_up,
                                       normal_down);
}
```

Energy calculations

```
#include <openbabel/forcefield.h>
#include <openbabel/mol.h>

OBMol mol;
/* ... */

// Select the MMFF94 forcefield
OBForceField* pFF;
pFF = OBForceField::FindForceField("MMFF94");
if (!pFF) exit(1);

// Set the logfile
pFF->SetLogFile(&std::cerr);

// Assign atom types, parameters, ...
pFF->Setup(mol);

// Calculate the energy
pFF->Energy();

// Perform maximum 1000 steps of minimization
pFF->ConjugateGradients(1000);
```

Open Babel

This documentation is part of the Open Babel project. For more information, see www.openbabel.org. Open Babel is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation version 2 of the License. Open Babel is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.

Spectrophore™

Spectrophore™ is a trademark of Silicos NV and the technology is part of the Open Babel project.

Silicos NV

Silicos is a fee-for-service company empowering open source chemo-informatics virtual screening technologies for the discovery of novel lead compounds and database characterization. Silicos fully endorses the concept of open innovation and open source software development, and provides its clients with a wide variety of computational chemistry-based lead discovery services, including Open Babel support, training and code development. Please visit www.silicos.com for more details.

Silicos NV
Wetenschapspark 7,
B-3590 Diepenbeek
Belgium

www.silicos.com
www.openbabel.org

www.silicos.com