RDKit: A software suite for cheminformatics, computational chemistry, and predictive modeling

Greg Landrum

glandrum@users.sourceforge.net

Last updated: Feb. 2011

Availability

- SourceForge Page:
  http://rdkit.sourceforge.net

- Source code:
  - Browse:
    http://svn.sourceforge.net/viewcvs.cgi/rdkit/trunk/
  - Download using subversion:
    svn co https://svn.sourceforge.net/svnroot/rdkit/trunk rdkit

- Binaries: available for Win32

- Licensing: BSD license (except for GUI components, which are GPL)

Components

End User

- Command-line scripts
- Python interface
- Database extensions

Developer

- Python programming library
- C++ programming library

General Molecular Functionality

- Input/Output: SMILES/SMARTS, mol, SDF, TDT
- “Cheminformatics”:
  - Substructure searching
  - Canonical SMILES
  - Chirality support
  - Chemical transformations
  - Chemical reactions
  - Molecular serialization (e.g. mol <-> text)
- 2D depiction, including constrained depiction and mimicking 3D coords
- 2D->3D conversion/conformational analysis via distance geometry
- UFF implementation for cleaning up structures
- Fingerprinting (Daylight-like, circular, atom pairs, topological torsions, “MACCS keys”, etc.)
- Similarity/diversity picking (include fuzzy similarity)
- 2D pharmacophores
- Gasteiger-Marsili charges
- Hierarchical subgraph/fragment analysis
- Hierarchical RECAP implementation

General Molecular Functionality, cntd

- Feature maps and feature-map vectors
- Shape-based similarity
- Molecule-molecule alignment
- Shape-based alignment (subshape alignment)*
- Very fast 3D pharmacophore searching
- Integration with PyMOL for 3D visualization
- Database cartridge

** functional implementations, but not really recommended for use*

General “QSAR” Functionality

- Molecular descriptor library:
  - Topological (κ3, Balaban J, etc.)
  - Electrotopological state (EState)
  - clogP, MR (Wildman and Crippen approach)
  - “MOE like” VSA descriptors
  - Feature-map vectors
- Machine Learning:
  - Clustering (hierarchical)
  - Information theory (Shannon entropy)
  - Decision trees, *naïve Bayes*, kNN**
  - Bagging, random forests
  - Infrastructure:
    - data splitting
    - shuffling (y scrambling)
    - out-of-bag classification
    - serializable models and descriptor calculators
    - enrichment plots, screening, etc.

** functional implementations, but not really recommended for use*

Reading/Writing Molecules

```
MolFromSmiles
MolFromSmarts
MolFromMolBlock
MolFromMolFile
```

```
mol = Chem.MolFromSmiles('CCCOCC')
```

```
SmilesMolSupplier
SDMolSupplier
TDTMolSupplier
SupplierFromFilename
```

```
mols = [x for x in Chem.SDMolSupplier('input.sdf')]
```

```
MolToSmiles
MolToSmarts
MolToMolBlock
```

```
print >>outF,Chem.MolToMolBlock(mol)
```

```
SmilesWriter
SDWriter
TDTWriter
```

```
w = Chem.SDWriter('out.sdf')
for m in mols:
   w.write(m)
```

Working with molecules: Substructure matching

```
>>> m = Chem.MolFromSmiles('O=CCC=O')
>>> p = Chem.MolFromSmarts('C=O')
>>> m.HasSubstructMatch(p)
True
>>> m.GetSubstructMatch(p)
(1, 0)
>>> m.GetSubstructMatches(p)
((1, 0), (3, 4))
>>> m = Chem.MolFromSmiles('C1CCC1C(=O)O')
>>> p= Chem.MolFromSmarts('C=O')
>>> m2 =Chem.DeleteSubstructs(m,p)
>>> Chem.MolToSmiles(m2)
'O.C1CCC1'
>>>
```

```
>>> def  findcore(m):
...   smi = Chem.MolToSmiles(m)
...   patt = Chem.MolFromSmarts('[D1]')
...   while 1:
...      m2 = Chem.DeleteSubstructs(m,patt)
...      nSmi = Chem.MolToSmiles(m2)
...      m = m2
...      if nSmi==smi:
...         break
...      else:
...         smi = nSmi
...   return m
...
>>> m = Chem.MolFromSmiles('CCC1CCC1')
>>> c = findcore(m)
>>> Chem.MolToSmiles(c)
'C1CCC1'
```

Working with molecules: properties

```
>>> suppl = Chem.SDMolSupplier('divscreen.400.sdf')
>>> m = suppl.next()
>>> list(m.GetPropNames())
['Formula', 'MolWeight', 'Mol_ID', 'Smiles', 'cdk2_ic50', 'cdk2_inhib',
'cdk_act_bin_1', 'mol_name', 'scaffold', 'sourcepool']
>>> m.GetProp('scaffold')
'Scaffold_00'
>>> m.GetProp('missing')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: 'missing'
>>> m.HasProp('missing')
0
>>> m.SetProp('testing','value')
>>> m.GetProp('testing')
'value'
>>> m.SetProp('calcd','45',computed=True)
>>> list(m.GetPropNames())
['Formula', 'MolWeight', 'Mol_ID', 'Smiles', 'cdk2_ic50', 'cdk2_inhib',
'cdk_act_bin_1', 'mol_name', 'scaffold', 'sourcepool', 'testing']
```

Generating Depictions

```
>>> from rdkit import Chem
>>> from rdkit.Chem import AllChem
>>> m = Chem.MolFromSmiles('C1CCC1')
>>> m.GetNumConformers()
0
>>> AllChem.Compute2DCoords(m)
0
>>> m.GetNumConformers()
1
>>> print Chem.MolToMolBlock(m)



  4  4  0  0  0  0  0  0  0  0999 V2000
    1.0607    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.0000   -1.0607    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -1.0607    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    1.0607    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0
  2  3  1  0
  3  4  1  0
  4  1  1  0
M  END

>>>
```

Generating 3D Coordinates

```
>>> from rdkit import Chem
>>> from rdkit.Chem import AllChem
>>> m = Chem.MolFromSmiles('C1CCC1')
>>> AllChem.EmbedMolecule(m)
0
>>> m.GetNumConformers()
1
>>> AllChem.UFFOptimizeMolecule(m)
0
>>> m.SetProp('_Name','testmol')
>>> print Chem.MolToMolBlock(m)
testmol


  4  4  0  0  0  0  0  0  0  0999 V2000
   -0.8040    0.5715   -0.2537 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.3727   -0.9165   -0.2471 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.7942   -0.5376    0.6386 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.3825    0.8826    0.6323 C   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0
  2  3  1  0
  3  4  1  0
  4  1  1  0
M  END
>>> list(AllChem.EmbedMultipleConfs(m,10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> m.GetNumConformers()
10
```

Low-budget conformational analysis

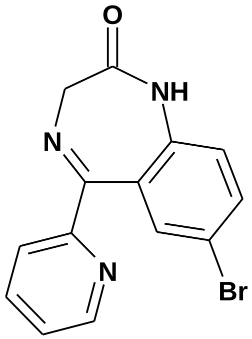
```
>>> from rdkit import Chem
>>> from rdkit.Chem import AllChem
>>> from rdkit.Chem import PyMol
>>> v = PyMol.MolViewer()

>>> m=Chem.MolFromSmiles('OC(=O)C1CCC(CC(=O)O)CC1')
>>> AllChem.EmbedMultipleConfs(m,10)
<rdBase._vectint object at 0x00A2D2B8>

>>> for i in range(m.GetNumConformers()):
...     AllChem.UFFOptimizeMolecule(m,confId=i)
...     v.ShowMol(m,confId=i,name='conf-%d'%i,showOnly=False)

>>> w = Chem.SDWriter('foo.sdf')
>>> for i in range(m.GetNumConformers()):
...     w.write(m,confId=i)
...
>>> w.flush()
```

Molecular Miscellany

```
>>> from rdkit import Chem
>>> from rdkit.Chem import Crippen
>>> Crippen.MolLogP(Chem.MolFromSmiles('c1ccccn1'))
1.0815999999999999
>>> Crippen.MolMR(Chem.MolFromSmiles('c1ccccn1'))
24.236999999999991

>>> AllChem.ShapeTanimotoDist(m1,m2)

>>> Chem.Kekulize(m)

>>> m = Chem.MolFromSmiles('F[C@H]([Cl])Br')
>>> Chem.AssignAtomChiralCodes(m)
>>> m.GetAtomWithIdx(1).GetProp('_CIPCode')
'R'

>>> m = Chem.MolFromSmiles(r'F\C=C/Cl')
>>> Chem.AssignBondStereoCodes(m)
>>> m.GetBondWithIdx(1).GetStereo()
Chem.rdchem.BondStereo.STEREOZ
```

Database CLI tools

```
# building a database:
% python $RDBASE/Projects/DbCLI/CreateDb.py --dbDir=bzr --molFormat=sdf bzr.sdf

# similarity searching:
% python $RDBASE/Projects/DbCLI/SearchDb.py --dbDir=bzr --molFormat=smiles \
  --similarityType=AtomPairs --topN=5 bzr.smi
[18:23:21] INFO: Reading query molecules and generating fingerprints
[18:23:21] INFO: Finding Neighbors
[18:23:21] INFO: The search took 0.1 seconds
[18:23:21] INFO: Creating output
Alprazolam,Alprazolam,1.000,Ro13-9868,0.918,Triazolam,0.897,Estazolam,0.871,U-35005,0.870
Bromazepam,Bromazepam,1.000,Ro05-3072,0.801,Ro05-3061,0.801,Nordazepam,0.801,Ro05-2921,0.772
 ...
Delorazepam,Delorazepam,1.000,Ro05-4619,0.900,Nordazepam,0.881,Lorazepam,0.855,Ro20-8065,0.840

# substructure and property searching:
% python $RDBASE/Projects/DbCLI/SearchDb.py --dbDir=bzr --smarts='c1ncccc1' \
  -q 'activity>6.5' --sdfOut=search1.sdf
[18:27:25] INFO: Doing substructure query
[18:27:25] INFO: Found 1 molecules matching the query
[18:27:25] INFO: Creating output
Bromazepam
[18:27:25] INFO: Done!
```



RDKit Developer's Overview

Greg Landrum

glandrum@users.sourceforge.net

Installation

- Download and install boost libraries (www.boost.org)
- Download distribution from http://rdkit.sourceforge.net (or get the latest version using subversion)
- Download and install other python dependencies (see $RDBASE/Docs/SoftwareRequirements.txt)
- Follow build instructions in $RDBASE/INSTALL

Documentation

- C++:

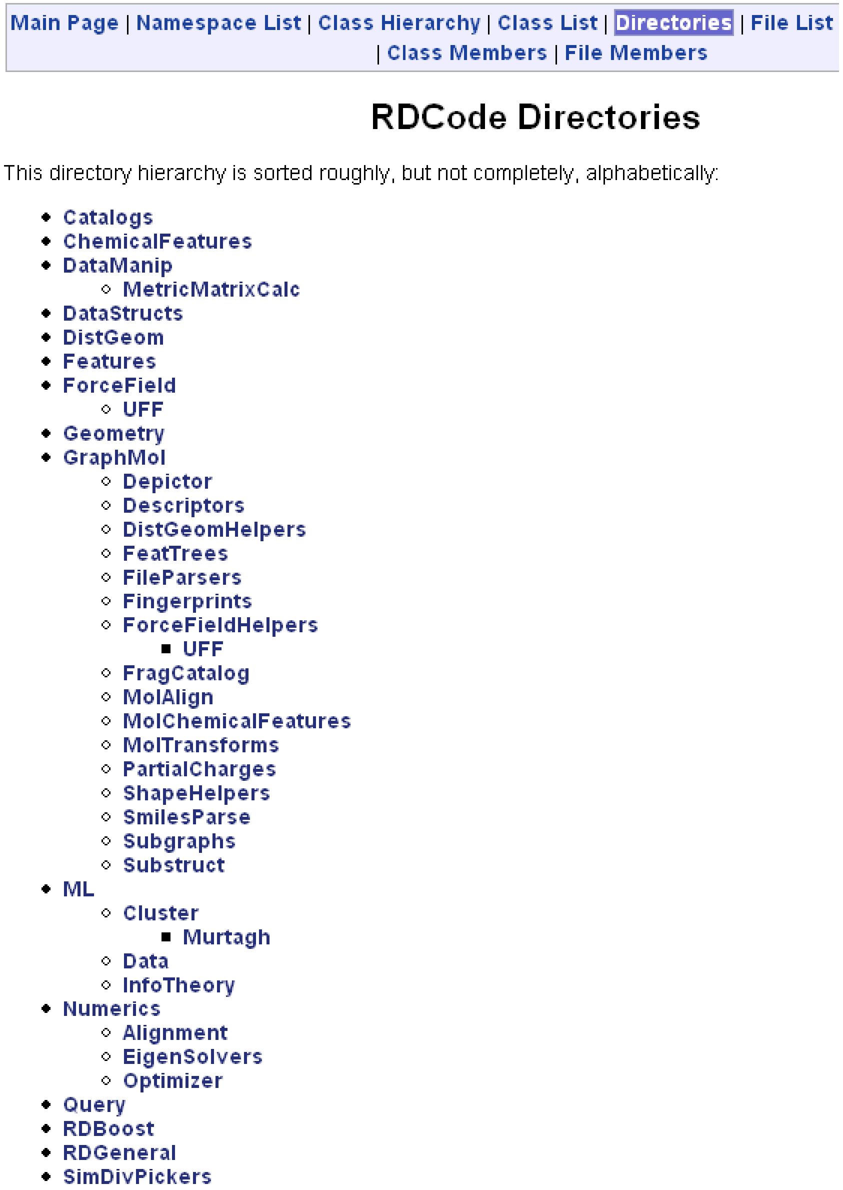
  Generated using doxygen:

  ```
  cd $RDBASE/Code
  doxygen doxygen.config
  ```

- Python:

  Currently generated using epydoc:

  ```
  cd $RDBASE/rdkit
  epydoc --config epydoc.config
  ```


Main Page | Namespace List | Class Hierarchy | Class List | Directories | File List | Class Members | File Members

RDCode Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

- Catalogs
- ChemicalFeatures
- DataManip
  - MetricMatrixCalc
- DataStructs
- DistGeom
- Features
- ForceField
  - UFF
- Geometry
- GraphMol
  - Depictor
  - Descriptors
  - DistGeomHelpers
  - FeatTrees
  - FileParsers
  - Fingerprints
  - ForceFieldHelpers
    - UFF
  - FragCatalog
  - MolAlign
  - MolChemicalFeatures
  - MolTransforms
  - PartialCharges
  - ShapeHelpers
  - SmilesParse
  - Subgraphs
  - Substruct
- ML
  - Cluster
    - Murtagh
  - Data
  - InfoTheory
- Numerics
  - Alignment
  - EigenSolvers
  - Optimizer
- Query
- RDBoost
- RDGeneral
- SimDivPickers

Organization

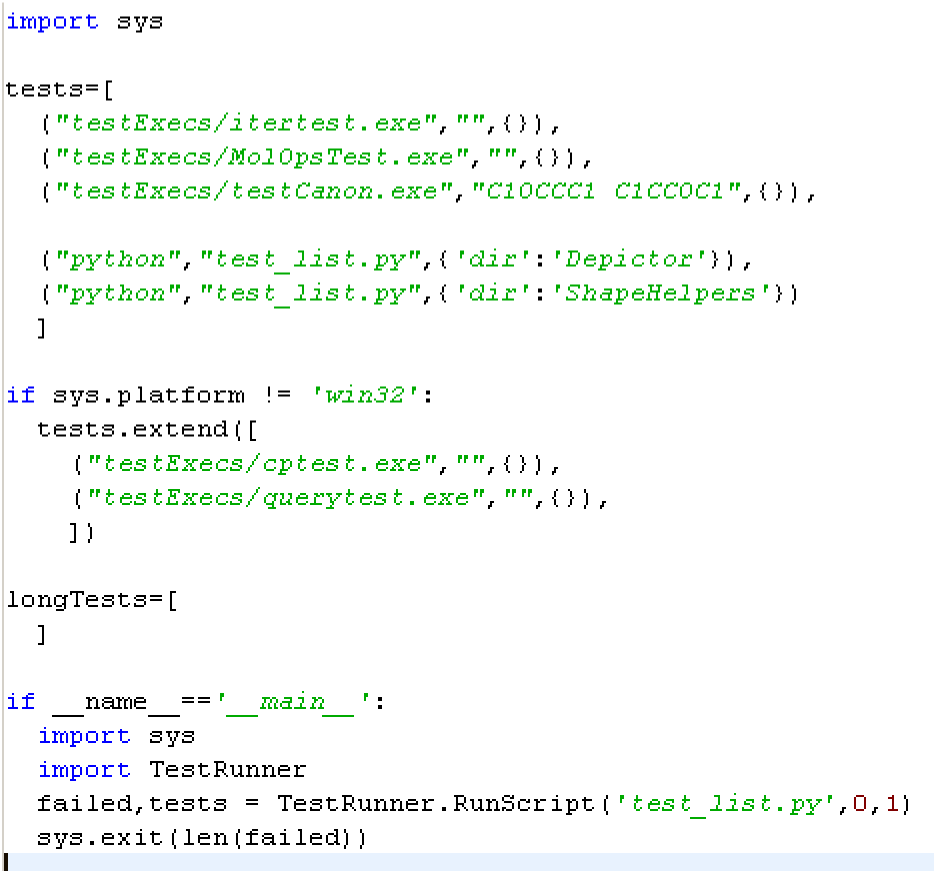
- $RDBASE/
  - External/: essential 3rd party components that are hard to get or have been modified
  - Code/: C++ library, wrapper, and testing code
  - rdkit/: Python library, GUI, and scripts
  - Docs/: Manually and automatically generated documentation
  - Data/: Testing and reference data
  - Scripts/: a couple of useful utility scripts
  - Web/: some simple CGI applications
  - bin/: installation dir for DLLs/shared libraries

C++ "Guiding Ideas"

- If boost already has the wheel, don't re-invent it.
- Try to keep classes lightweight
- Maintain const-correctness
- Test, test, test
- Include tests with code
- Keep namespaces explicit (i.e. no `using namespace std;`)
- Keep most everything in namespace `RDKit`
- Test, test, test

Test Harness

Sample test_list.py:



```
import sys

tests=[
  ("testExecs/itertest.exe","",{}),
  ("testExecs/MolOpsTest.exe","",{}),
  ("testExecs/testCanon.exe","C1OCCC1 C1CCOC1",{}),

  ("python","test_list.py",{'dir':'Depictor'}),
  ("python","test_list.py",{'dir':'ShapeHelpers'})
  ]

if sys.platform != 'win32':
  tests.extend([
    ("testExecs/cptest.exe","",{}),
    ("testExecs/querytest.exe","",{}),
    ])

longTests=[
  ]

if __name__=='__main__':
  import sys
  import TestRunner
  failed,tests = TestRunner.RunScript('test_list.py',0,1)
  sys.exit(len(failed))
```

Wrapping code using BPL: Intro

- boost::python is not an automatic wrapper generator: wrappers must be written by hand.
- very flexible handling of "primitive types"
- tight python type integration
- allows subclassing of C++ extension classes

Wrapping code using BPL: defining a module

DataStructs/Wrap/DataStructs.cpp

```
#include <boost/python.hpp>
#include <RDBoost/Wrap.h>
#include "DataStructs.h"

namespace python = boost::python;

void wrap_SBV();
void wrap_EBV();
...
BOOST_PYTHON_MODULE(cDataStructs)
{
  python::scope().attr("__doc__") =
    "Module containing an assortment of functionality for basic data structures.\n"
    "\n"
    "At the moment the data structures defined are:\n"
...
    ;
  python::register_exception_translator<IndexErrorException>(&translate_index_error);
  python::register_exception_translator<ValueErrorException>(&translate_value_error);
...
  wrap_SBV();
  wrap_EBV();
...
}
```

Wrapping code using BPL: defining a class

DataStructs/Wrap/wrap_SparseBV.cpp

```
struct SBV_wrapper {
  static void wrap(){
    python::class_<SparseBitVect>("SparseBitVect",
                                  "Class documentation",
                                  python::init<unsigned int>())
      .def(python::init<std::string>())
      .def("SetBit",(bool (SBV::*)(unsigned int))&SBV::SetBit,
           "Turns on a particular bit on.  Returns the original state of the bit.\n")
...
      .def("GetNumBits",&SBV::GetNumBits,
           "Returns the number of bits in the vector (the vector's size).\n")
};

void wrap_SBV() {
  SBV_wrapper::wrap();
}
```

Wrapping code using BPL: making it "pythonic"

DataStructs/Wrap/wrap_SparseBV.cpp

```
struct SBV_wrapper {
  static void wrap(){
    python::class_<SparseBitVect>("SparseBitVect",
                                  "Class documentation",
                                  python::init<unsigned int>())
...
      .def("__len__",&SBV::GetNumBits)
...
      .def("__getitem__",
             (const int (*)(const SBV&,unsigned int))get_VectItem)
      .def("__setitem__",
             (const int (*)(SBV&,unsigned int,int))set_VectItem)
...
      .def(python::self & python::self)
      .def(python::self | python::self)
      .def(python::self ^ python::self)
      .def(~python::self)
...
      ;
  }
};
```

Wrapping code using BPL: supporting pickling

DataStructs/Wrap/wrap_SparseBV.cpp

```
// allows BitVects to be pickled
struct sbv_pickle_suite : python::pickle_suite
{
  static python::tuple
  getinitargs(const SparseBitVect& self)
  {
    return python::make_tuple(self.ToString());
  };
};

struct SBV_wrapper {
  static void wrap(){
    python::class_<SparseBitVect>("SparseBitVect",
                                  "Class documentation",
                                  python::init<unsigned int>())
...
      .def_pickle(sbv_pickle_suite())
      ;
  }
};
```

Wrapping code using BPL: returning complex types

DataStructs/Wrap/wrap_SparseBV.cpp

```
struct SBV_wrapper {
  static void wrap(){
    python::class_<SparseBitVect>("SparseBitVect",
                                  "Class documentation",
                                  python::init<unsigned int>())
...
      .def("GetOnBits",
             (IntVect (*)(const SBV&))GetOnBits,
             "Returns a tuple containing IDs of the on bits.\n")
...
      ;
  }
};
```

- IntVect is a std::vector<int>
- Convertors provided for: `std::vector<int>, std::vector<unsigned>, std::vector<string>, std::vector<double>, std::vector< std::vector<int> >, std::vector< std::vector<unsigned> >, std::vector< std::vector<double> >, std::list<int>, std::list< std::vector<int> >` in rdBase.dll
- Others can be created using:

```
RegisterVectorConverter<RDKit::Atom*>();
RegisterListConverter<RDKit::Atom*>();
```

Wrapping code using BPL: accepting Python types

DataStructs/Wrap/wrap_SparseBV.cpp

```
void SetBitsFromList(SparseBitVect *bv, python::object onBitList) {
  PySequenceHolder<int> bitL(onBitList);
  for (unsigned int i = 0; i < bitL.size(); i++) {
    bv->SetBit(bitL[i]);
  }
}

struct SBV_wrapper {
  static void wrap(){
    python::class_<SparseBitVect>("SparseBitVect",
                                  "Class documentation",
                                  python::init<unsigned int>())
...
          .def("SetBitsFromList",SetBitsFromList,
               "Turns on a set of bits.  The argument should be a tuple or list of bit
ids.\n")
...
      ;
  }
};
```

Wrapping code using BPL: keyword arguments

GraphMol/Wrap/rdmolfiles.cpp

```
 docString="Returns the a Mol block for a molecule\n\
  ARGUMENTS:\n\
\n\
    - mol: the molecule\n\
    - includeStereo: (optional) toggles inclusion of stereochemical\n\
                     information in the output\n\
    - confId: (optional) selects which conformation to output (-1 = default)\n\
\n\
  RETURNS:\n\
\n\
    a string\n\
\n";
  python::def("MolToMolBlock",RDKit::MolToMolBlock,
      (python::arg("mol"),python::arg("includeStereo")=false,
       python::arg("confId")=-1),
       docString.c_str());
```

This also demonstrates defining a function and taking a complex (wrapped) type as an argument

Wrapping code using BPL: returning pointers

GraphMol/Wrap/rdmolfiles.cpp

```
   docString="Construct a molecule from a SMILES string.\n\n\
  ARGUMENTS:\n\
    - SMILES: the smiles string\n\
    - sanitize: (optional) toggles sanitization of the molecule.\n\
      Defaults to 1.\n\
  RETURNS:\n\
    a Mol object, None on failure.\n\
\n";
  python::def("MolFromSmiles",RDKit::MolFromSmiles,
      (python::arg("SMILES"),
       python::arg("sanitize")=true),
      docString.c_str(),
      python::return_value_policy<python::manage_new_object>());
```

GraphMol/Wrap/Mol.cpp

```
.def("GetAtomWithIdx",(ROMol::GRAPH_NODE_TYPE (ROMol::*)(unsigned int))
                       &ROMol::getAtomWithIdx,
 python::return_value_policy<python::reference_existing_object>(),
 "Returns a particular Atom.\n\n"
 "  ARGUMENTS:\n"
 "    - idx: which Atom to return\n\n"
 "  NOTE: atom indices start at 0\n")
```