Share Documentation

Jeff Martin

April 13, 2009

1 Introduction

The Share framework is intended to be a library of general APIs and codes to perform mundane tasks related to protein modeling. To the author's knowledge, no general freely-available APIs implemented in Java exist to perform these tasks. To be fair, the implementations provided in this project are in many clases simplistic (and in a few cases incomplete), but they perform sufficiently well to support the needs of the JDSHOT project when working with p53 - the only project currently built on top of Share. Also, the author does not suggest that the implementations of functions/classes here are necessarily the best way to do it and would be happy to hear suggestions for improvement.

It is the author's goal that this framework be (as the name might imply) shared with developers interested in protein modeling to provide a basis for starting new projects quickly.

2 Packages

All packages are under the root package/namespace: edu.duke.donaldLab.share

(root) The root package houses the main function for unit testing

atomType Code for specifying types for atoms (e.g., C.1, C.ar, N.1, N.3) and van der Waal radii. This is configured using the `.atomTypes` resource files.

bond Code for specifying bonding structure for sets of atoms. Bonds are represented as edges in a graph and can be configured per amino acid and backbone state (i.e. normal vs terminus) using the `.bonds` resource files. Performing BFS in this graph is also supported using iterators. This package is incomplete (not all amino acids are defined), but is sufficient for p53.

clustering This code provides methods to cluster sets of points using an implicit Euclidean distance metric as well as using a distance matrix. The Euclidean method is rather sophisticated and fast (!), but the distance matrix method is currently very naive and extremely slow.

geom These classes represent generic geometric objects such as boxes and spheres and can perform intersection tests between pairs of these objects. Also implemented is an algorithm that computes the min bounding sphere for a set of points.

io The command-line argument processor is implemented here along with a few extra classes to operate on streams.

kinemage This package provides a simplistic model of the kinemage format and provides an API for a subset of its features. Also included is code to render geometric objects (e.g., proteins, NOEs) into the kinemage format.

mapping Classes here perform various mappings between various namespaces. For example, this code can map between a couple atom naming conventions found in PDB and MR files. It also maps atom addresses between text formats (e.g., `resid 36 and name HE22 and segid A`) and a more efficient binary format. In particular, the atom naming convention mapping is incomplete (but sufficient for p53), but can be adjusted/improved by modifying the `.map` resource files.

math The math package contains vector libraries in 3 and higher dimensions, matrix operations in 3 and 4 dimensions, and higher-dimensional analogues to the objects in the geom package. Implementations for quaternions are also included along with a class that computes optimal rotations between two point sets under an RMSD objective function.

minimize Wrapper classes for the CNS integration

mol2 For now, contains a single class that exports proteins in the MOL2 format.

nmr Contains code for reading/writing NOEs from MR files, mapping these NOEs to a protein in memory, and a class that removes all atoms from a protein object except for those referenced by NOEs (useful as an optimization).

pdb Contains code to read/write PDB files into/from protein memory objects. Also has a class to update PDB files written by CNS to the more modern format.

perf This package contains timers and progress bars.

protein This package contains all the classes needed to represent a protein in memory. Protein objects in memory are organized in a heirarchical structure as follows: A protein can contain many subunits. A subunit can contain many residues. A residue can contain many atoms. Finally, an atom can contain many models. Each model for an atom is essentially a 3D point specifying the atom's position. Also, two schemes exist for addressing atoms in a protein object. The readable format (`ReadableAtomAddresser`) is human-readable but slow to process. There is also binary format (`AtomAddresser`) that is not human readable, but is much faster.

pseudoatoms Implementation of pseudoatoms for proteins. This package is configured using a resource file `pseudo.atoms`.

steric Performs steric checking for protein objects.

test This package contains the unit tests

util Miscellaneous code

3 Resources

Resources are "extra" files included in the program distribution that are used during execution (e.g., configuration, settings, etc). These files get baked into the JARs under the package *resources* with the exception of the files under *resources.test*. Testing files are excluded from all distribution files. When writing code to read these files, one must use methods in the class loader to get an input stream for the resource. Simply opening a file at that path will only work in development. In the deployed setting (when the application is executed from JAR files), the paths will not be available. Example code to load a resource file is as follows:

```
InputStream in = getClass().getResourceAsStream( "/resources/myResource.file" );
```

Note that `getClass()` might not work in all settings. However, any reference to an object of type `Class` is sufficient.

4 Testing

Wherever possible, code is written such that it allows unit testing to ensure component correctness. In a nutshell, unit testing is the practice of writing a little extra code (unit tests) to execute application code and verify that the result is correct. The unit tests for this project are implemented using the JUnit automated testing framework.

The code is also written using the built-in assert library for java. However, the asserts are not enabled at runtime by default. When running code for debugging purposes, it is useful to turn on the asserts to help trap errors. This can be done by using the java VM argument `-ea`.