MIT Java Wordnet Interface 1.1: User's Guide

Mark A. Finlayson

(markaf@mit.edu)

1. Purpose of the Software

This MIT Java Wordnet Interface (JWI) was designed to be an easy-to-use, easy-to-extend Java library for interfacing with the Wordnet electronic dictionary. It features API calls to retrieve index words, synsets, and morphological exceptions from the Wordnet data files. It also has calls that allow browsing by following synset pointers, and classes that can perform simple morphological processing. The library includes no GUI elements.

Version 1.1 of JWI is compatible with Wordnet versions 2.1 and 3.0. Wordnet is **not** included with any JWI distribution; Wordnet must be downloaded separately from the Wordnet site at http://wordnet.princeton.edu.

2. Simple Example

The main interface for accessing dictionary data is the *IDictionary* interface in the *edu.mit.jwi.dict* package. The distribution comes with a single default implementation of this interface in the same package, the *Dictionary* class. In the simplest case, where you are using Wordnet with the data files on the same filesystem as your Java program, you can instantiate the *Dictionary* class with a single argument, a Java *URL* object that points to the directory where the Wordnet dictionary data files are located.

An example of this can be found in Figure 1, in the form of a Java method *testDictionary()*. In that method, the first block of seven lines (4-10) deals with constructing a *URL* object that points to the Wordnet data files. In this particular example, the base Wordnet directory is assumed to be stored in a system environment variable called **WNHOME**; this may be different on your system. Note that the **WNHOME** variable points to the root of the Wordnet installation directory and the dictionary data directory "dict" must be appended to this path. Again, this may be different on your system depending on where your Wordnet files are located. The second block of code, two lines long (13-14), constructs an instance of the default *Dictionary* object, and opens it by calling the *open()* method. The final block of six lines (17-22) demonstrates searching the dictionary for the first sense of the noun "dog". Figure 2 shows the console output of the method.

```
public void testDictionary(){

   // construct the URL to the wordnet dictionary directory
   String wnhome = System.getenv("WNHOME");
   String separator = System.getProperty("file.separator");
   String path = wnhome + separator + "dict";
   URL url = null;
   try{ url = new URL("file", null, path); }
   catch(MalformedURLException e){ e.printStackTrace(); }
   if(url == null) return;

   // construct the dictionary object and open it
   IDictionary dict = new Dictionary(url);
   dict.open();

   // look up first sense of the word "dog"
   IIndexWord idxWord = dict.getIndexWord("dog", PartOfSpeech.NOUN);
   IWordID wordID = idxWord.getWordIDs()[0];
   IWord word = dict.getWord(wordID);
   System.out.println("Id = " + wordID);
   System.out.println("Lemma = " + word.getLemma());
   System.out.println("Gloss = " + word.getGloss());

}
```

Figure 1: Sample Dictionary code

```
Id = WID-2001223-n-?-dog
Lemma = dog
Gloss = a member of the genus Canis (probably descended from the
common wolf) that has been domesticated by man since prehistoric
times; occurs in many breeds; "the dog barked all night"
```

Figure 2: Sample Dictionary Code Output (for Wordnet 2.1)

3. Modifying and Extending the Software

One design goal of this software library was for it to be easy to extend. To that end, most classes are governed by a corresponding Java interface, and they are manipulated as such in the code. Thus, to add your own functionality, you must only find the proper insertion point for your code and implement the appropriate interfaces.

The main interfaces for customizing dictionary behavior are the *IDataProvider* and *IParserProvider* interfaces, instances of which can be passed to a constructor for the default *Dictionary* class. These objects control the way the dictionary accesses the data in the Wordnet data files. For those more adventurous, the *IDictionary* interface can be re-implemented from the ground up.

4. Planned Features

The following features are planned for future versions. Listed in order of importance:

1. Provide access to sense count files
2. Provide JUnit test-suite for development purposes

Unfortunately I am monolingual, and therefore do not plan to personally add support for international Wordnets (for example, the Italian wordnet or Euro Wordnet). If there is an enterprising coder that wishes to tackle this project, I will lend whatever support I can.

If you find a bug in the software, wish to request a feature, or submit extensions (with appropriate acknowledgment) to be included in a future release, please contact me via my permanent email forwarding address markaf@alum.mit.edu.