Java Source Reflection

1. Why Source Reflection?

Java Source Generation is frequently based on reflection. For example, the Proxy Generator works roughly like this: Class A is inspected using Java Reflection. A new class B is created. For any public method in A, a method in B is created, that invokes A.

This approach has a subtle inconvenience: To use Java reflection, the class A must have already been compiled. As a consequence, the use of generated sources typically happens in three stages:

1. Compiling a part of the sources
2. Invoking the source generator
3. Compiling the remaining sources

This can become rather nasty. In particular, you cannot reference the second part of the sources from the first part. The build scripts tend to be overly complex and difficult to maintain.

Java Source Reflection is a true simplification of the above process: Required informations are gathered from the Java source files, and not from the compiled classes.

2. How it works

Java Source Reflection is implemented by the class JavaParser. This class takes as input a Java source file and converts it into an instance of JavaSource. The Java parser is internally based on an AntLR parser. (AntLR is a public domain parser generator.)

The created JavaSource instance contains instances of JavaMethod, JavaField, JavaInnerClass, and so on. Obviously, these can be used to replace classical Java reflection.

3. Using the JavaParser

To use the Java parser, your classpath must obviously contain the file jaxmejs.jar. However, because the actual parser is generated by AntLR, you need the file antlr.jar as well. Both files come to you as part of the JaxMe distribution. Besides, a current version of antlr.jar (2.7.4, as of this writing) can always be obtained from www.antlr.org. However, if you replace the AntLR parser, then you should probably use the JaxMe source distribution, and rebuild the

binaries, thus creating a new parser.

The following sample uses Java source reflection to print all public non-static methods of a certain Java class:

```
import org.apache.ws.jaxme.js.*;

public void printPublicInstanceMethods(File pFile) {
  JavaSourceFactory jsf = new JavaSourceFactory();
  JavaParser jp = new JavaParser(jsf);
  jp.parse(pFile);
  for (Iterator iter = jsf.getJavaSources();  iter.hasNext();  ) {
    JavaSource js = (JavaSource) iter.next();
    System.out.println("Public instance methods of class: " + js.getQName());
    JavaMethod[] methods = js.getMethods();
    for (int i = 0;  i < methods.length;  i++) {
      if (methods[i].getProtection().equals(JavaSource.PUBLIC)  &&
          !methods[i].isPublic()) {
        System.out.println("  " + methods[i].getName());
      }
    }
    System.out.println(js.getQName());
  }
}
```