Creating custom tool

There are a few steps involved in developing a custom tool. These steps can be summarized as follows:

1. Developing a Tool Class
2. Creating an Extension Package

Developing a Tool Class

Developing a Tool class requires setting a class path to *magicreport.jar*. You can find *magicreport.jar* from the library folder under the Report Wizard plugin directory. The following sample shows the source code for *Hello.java*.

```
package mytool;
import com.nomagic.magicreport.engine.Tool;
public class HelloTool extends Tool
{
   public String getHello() {
      return "Hello World";
   }
   public String getHello(String name) {
      return "Hello " + name;
   }
}
```

The sample shows two methods following the getter concept defined by the JavaBean specification.

Creating an Extension Package

The extension package is delivered in a *JAR* file. *JAR* (Java ARchive) is a file format based on the popular *ZIP* file format and is used for aggregating many files into one. To create a *JAR* file, you can store *.class with the Java package folder structure in a *ZIP* file format or creating from a *JAR* tool.

To combine files into a *JAR* file (in this case), here is the sample code fragment:

```
jar cf MyTool.jar *.class
```

In this example, all the class files in the current directory are placed in the file named "*MyTool.jar*". A manifest file entry named *META-INF/MANIFEST.MF* is automatically generated by the *JAR* tool. The manifest file is the place where any meta-information about the archive is stored as named: Value pairs. Please refer to the *JAR* file specification for more details.

```
jar cf MyTool.jar mytool
```

The example above shows that all the class files in the directory mytool are placed in the file named "*MyTool.jar*".

A complete *JAR* tool tutorial can be found at:

- http://java.sun.com/docs/books/tutorial/deployment/jar/
- http://java.sun.com/javase/6/docs/technotes/tools/windows/jar.html