JxRef Documentation

Table of contents

1. Overview

JxRef is a Java code analysis toolset. It is a tool that will help you identify refactoring opportunities in a Java code base. It can also provide information that will help you identify potential risks of making a code change.

JxRef will do the following:

- Identify if a class or method is used, If so, where..
- Identify Activity Flow information about a method (what it calls in what classes)
- Identify application dependencies (on open source projects, for example).
- Identify the largest and most complex methods (according to McCabe Complexity metrics)
- Identify methods that are unused or should be private

2. Installing JxRef

Insure that Apache Ant (http://ant.apache.org/) is installed and properly configured.

Unzip the distribution into a directory of your choosing. The distribution explicitly contains a directory with name and version information. For example, if you unzip the V1.0 distribution to `/opt/tools`, JxRef will be located at `/opt/tools/jxref-1.0`.

Define environment variable JXREF_HOME to be the directory that contains the JxRef install (e.g. `/opt/tools/jxref-1.0`). By the way, setting the JXREF_HOME environment variable is required for JxRef to work properly.

3. Running JxRef Reports

All JxRef reports are run from Ant (http://ant.apache.org/) . In order for your Ant project to have access to JxRef tasks, you need to add the following to lines to your ant script at the project level:

```
<property environment="env"/>
<import file="${env.JXREF_HOME}/jxref-ant.xml"/>
```

The imported ant script will append everything you need to the class path and define JxRef tasks, so you don't have to. Note that the `env` label is arbitrary, any label will do as long as it's consistent with what's in the import statement.

Sample Ant scripts can be found in `$JXREF_HOME/samples`.

All JxRef reports are run using the `jxref` task. Supported options for the jxref task are as

follows:

- · **`outDir`** -- *(Optional)* the name of the output directory in which to place the reports.
- · **`format`** -- *(Optional)*Values are `xml` and `html` – both formats are produced by default
- · **`label`** -- *(Optional)*Specifies a label to put on the top of each report. For example, 'The Custom Application'

Supported subtags for the jxref task are as follows:

- · **`fileset`** -- *(Required)* Describes the set of class files to be analyzed. See Ant documentation for fileset options.
- · **`include`** -- *(Optional)*Specifies what is included in report output by className, methodName, and methodModifier. see details below.
- · **`exclude`** -- *(Optional)* Specifies what is excluded in report output by className, methodName, and methodModifier. See details below.
- · **`methodXref`** -- *(Optional)* Method Cross-Reference report. Provides a list of which classes reference each method. See section below for details.
- ·**`unusedMethod`** -- *(Optional)* Unused Method report. Provides lists of methods that aren't used and methods that could potentially be made `private`. See section below for details.
- · **`sequenceFlow`** -- *(Optional)*Sequence Flow Report. Provides information you would normally find on a UML sequence diagram. See section below for details.
- ·**`complexity`** -- *(Optional)*Complexity Report. Provides size and complexity metrics. See section below for details.
- ·**`unusedClass`** -- *(Optional)*UnUsed Class Report. Provides a list of classes which do not appear to be directly utilized by the analyzed code base

The include/exclude tags are available at a report level as well. Some attributes support the '*' wild card character, which acts the same way it does using the dir or ls commands at your operating system prompt. These tags support the following attributes:

Tag Name	Attribute	Description
`include`	**`className`**	Mask describing classes *included* in report output. Note that class names are fully qualified (e.g. `java.lang.System`). The * wild card character is permitted.
	`methodName`	Mask describing methods *included* in report output. The * wild card character is permitted.
	`methodModifier`	Describes modifiers of reported

		methods. Allowed values are: `public, private, protected, static, abstract, final, interface, strict, synchronized, transient,` and `volatile`.
`exclude`	**`className`**	Mask describing classes *excluded* in report output. Note that class names are fully qualified (e.g. `java.lang.System`). The * wild card character is permitted.
	`methodName`	Mask describing methods *excluded* in report output. The * wild card character is permitted.
	`methodModifier`	Describes modifiers of reported methods. Allowed values are: `public, private, protected, static, abstract, final, interface, strict, synchronized, transient,` and `volatile`.

3.1. Example Ant Script

The sample illustrates the verbage above. It is included in the `JXREF_HOME/samples` directory.

```
<project default="samples" basedir=".">

  <!-- Include these two lines in any antscript to use JxRef tasks-->
  <property environment="env"/>
  <import file="${env.JXREF_HOME}/jxref-ant.xml"/>

        <!-- Generate sample reports -->
        <target name="samples" >
                <jxref format="xml" label="This is a sample label">
                  <fileset dir="../lib">
                                <include name="**/*.jar"/>
     </fileset>
                        <methodXref>
```

```
                        <exclude className="java.*"/>
                        <exclude className="javax.*"/>
                    </methodXref>
                    <unusedMethod/>
                    <sequenceFlow/>
                    <complexity/>
                </jxref>
        </target>

</project>
```

4. JxRef Report Definitions and Samples

4.1. Method Cross Reference Report

This report will list all methods in your application and the methods that call them. Note that the `methodModifier` options on `include` and `exclude` tags are not supported for this report.

See sample HTML (jxref-methodXref.html) and XML (jxref-methodXref.xml) reports.

4.2. Unused Method Report

This report will list all methods that are unused or can be made private.

See sample HTML (jxref-unusedMethod.html) and XML (jxref-unusedMethod.xml) reports.

4.3. Sequence Flow Report

This report will list information that would be found in a UML sequence diagram.

See sample HTML (jxref-sequenceFlow.html) and XML (jxref-sequenceFlow.xml) reports.

- · **`maxCallDepth`** -- *(Optional)* Describes how many levels of method calls are analyzed. This number must be greater than zero. The higher the depth, the higher your memory needs will be. The default is 5.

4.4. Complexity Report

This report will provide complexity metrics that will highlight the most complex portions (and probably the most difficult portions to maintain) of the analyzed code base. You can use this report to target refactoring efforts.

See sample HTML (jxref-complexity.html) and XML (jxref-complexity.xml) reports.

- · **`top`** -- *(Optional)* Describes how many methods appear in the 'most-complex-methods'

section. The default is 20.

4.5. Uncalled classes Report

This report will provide a list of classes that do not appeared to be used by the analyzed code base. Keep in mind that classes on this report may in fact be used *outside* the analyzed code base (e.g. published APIs) or dynamically via reflection. JxRef cannot identify reflexive usage.

5. JavaDoc Documentation

JavaDoc can be found here (api/index.html) and in the downloaded distribution.