RecoderAnalyser (Java)

Technical Background

29th of June, 2011

Written by:

Johan van der Geest

Mark Ettema

University of Groningen

Contents

1 Introduction 3
2 Using the command-line 4
2.1 Using information 4
2.2 Progress information 4
3 Options 5
3.1 Analyse 5
3.2 Merge 5
3.3 Remove 5
3.4 SolidSX2 dataset 5
4 RecoderAnalyser tree 6
4.1 SolidSXNode 6
4.1.1 SolidSXRootNode 6
4.1.2 SolidSXPackageNode 7
4.1.3 SolidSXClassNode 7
4.1.4 SolidSXFieldNode 7
4.1.5 SolidSXMethodNode 7
4.1.6 SolidSXConstructorNode 7
4.2 SolidSXReference 7

1 Introduction

RecoderAnalyser is used for the code analyzing part of the Recoder Project. The tool is written in Java because it uses Recoder (http://recoder.sourceforge.net) for the code analyzing. The results is serialized tree with dependencies of several revisions or an SolidSX2 dataset file.

In the Revision Analyser project the tool is integrated using the RecoderAnalyser C# class. It is also possible to use the tool directly from the command-line. This document describes the features and technical details of the tool.

In chapter 2 the command-line usage is described shortly. Chapter 3 give more details about the possible options and how they are implemented. Finally in chapter 4 the RecoderAnalyser tree is described, this tree is used to store dependency and revision information.

2 Using the command-line

The functions of RecoderAnalyser can be called using the command-line. This chapter describes the usages and progress information.

2.1 Using information

```
usage: RecoderProject <options>
  -analyse <sourcePath>        Analyse the JAVA source code from the given folder
  -ignore                      Ignore analyse errors, show warnings
  -input <bin file>            Input bin file (only needed when analyse parameter
                                is not used)
  -libfiles <lib files>        Add jar/.class files, use ; to split multiple files
  -libfolders <lib folder>     Add all .jar/.class files from a folder (including
                               subfolders), use ; to split multiple folders
  -merge <bin file>            Merge the result with the .bin file
  -output <outputPath>         Output path for the resulting bin file
  -remove <revision number>    Remove a revision from the input bin file (this
                                option is
                               ignored when using merge)
  -revision <revision number>  Revision number of the input source
  -solidsx2db <DB path>        Generate a SolidSX2 database file from the result
  -solidsx2xml <XML path>      Generate a SolidSX2 XML file from the result
```

To parse the given command-line options we used the Apache Common CLI package (http://commons.apache.org/cli/index.html). When there is a parse exception, a message with information about the exception and this usage information is shown.

2.2 Progress information

To give progress and error information we used the outputstream and the errorstream. This information can be easily parsed by other tools, in our case the C# class.
The outputstream is used to print progress messages and values. Each message has the following structure “STATUS_MESSAGE: *<message>*“ where *<message>* is replace with the right information. There is one exception to the rule: the message to indicate that program is finished is indicated by “STATUS_EXIT: *<exit-info>*” where *<exit-info>* is replace with information about how many errors are occurred during the process.

For some processes there is more detailed progress information available, this are values that indicate progress. This information can be used to show the progress with a progress-bar. Each value has the following structure: “STATUS_VALUE: *<value>*%” where *<value>* is replaced with the actual value in the range from zero to 100.

The error stream is used to print errors and warnings. Each error has the following structure: “ERROR_*<num>: <message>*” where *<num>* is replaced with the actual error number and *<message>* is replace with the information that belongs to the error.

3 Options

This chapter describes the options of RecoderAnalyser and gives some more details about the implementation of the tool.

3.1 Analyse

The “-analyse” option, with the path to a source folder as argument, is used to create a new RecoderAnalyser tree. With the “-revision” option you specify the revision number of the input source. When there are analyse errors, there is no tree generated. These errors can occur because there are reference found to not included .jar or class files. There are two possibilities to solve the problems: Add the necessary files using the “-libfiles” or “-libfolders” option is the best solution. But you can also ignore the errors with the “-ignore” option, in that case not all references will be shown in the tree.

When using the analyse option, the “-output” option should also be set, because this is the path to the folder or the file where the serialized result of the tree will be stored.

3.2 Merge

The “-merge” option, with the path to a RecoderAnalyser tree as argument, is used to merge two trees. Each node in the tree has a list of revision numbers. When a node from the first tree is equal to the node from the second tree, only the revision number is added. Otherwise, when the nodes are not equal, the node is added to the tree.

The “-output” option should be set, so the result is written to the right location. For the input tree the “-input” option can be used, in that case a serialized tree will be loaded. It is also possible to combine the analyze option with the merge option. In that case the analyse result is directly merged with the given tree.

3.3 Remove

The “-remove” option, with a revision number as argument, is used to remove a revision from a tree. The “-input” argument should be set, because a valid RevisionAnalyser tree should be loaded. The “-output” argument is optional, when this argument is not defined input file is updated with the resulting tree.

3.4 SolidSX2 dataset

To visualize a generated RecoderAnalyser tree, it should be converted to a dataset that is readable for SolidSX2. This can be done using the “-solidsx2xml” and “-solidsx2db” options. The result is a XML file or a database file. The database file is recommended because SolidSX2 can load database files faster. The export functions are classes with as input a RecoderAnalyer tree, so new export types can easily added. For example a SolidSX3 class.

4 RecoderAnalyser tree

The RecoderAnalyser tree is used to store the analyzed result. The tree is build with SolidSXNodes. There are different SolidSXNode types, but each type extends from the SolidSXNode class. Each class in the tree can be serialized, so the complete tree can be easily saved to file. This chapter describes the classes.

4.1 SolidSXNode

A SolidSXNode is the base for the several node types. It includes the following public methods:

- *getID:* Returns the node ID.
- *setID:* Set the node ID.
- *getName:* Returns the node name.
- *getParent:* Returns the parent node.
- *getType:* Returns the nodeType.
- *setParent:* Set the parent node.
- *getChildren:* Returns a list with child nodes.
- *getReferences:* Returns a list with references.
- *getRevisions:* Returns a list of revisions were the node is part of.
- *hasRevisions:* Returns true if the node is part of the given revision, otherwise false.
- *getChildByName:* Returns the childNode specified by the given name, otherwise null.
- *getChildByName:* Returns the child node specified by the given name and revision, otherwise null.
- *addChildNode:* Add the given node to the child list.
- *addReference:* Add the given SolidSXReference to the reference list.
- *addRevision:* Add the given revision number to the revision list.
- *sortChilderen:* Sort the child list by type and then by name.

4.1.1 SolidSXRootNode

A SolidSXRootNode is a SolidSXNode with as type “root”. The SolidSXRootNode is the top of the tree so it has no parent (=null). It includes the following additional public methods:

- *getNodeByPath:* This method returns a SolidSXNode at the given path, with the given revision. Null is returned when the node not exists.
- *nodeExists:* The same input as getNodeByPath, but the results is true if the node is found and false if the node not exists.
- *getRevision:* When the treee is not merged and has only one revision, this function returns the revision number. For a merged tree -1 is returned.
- *getStringRevision:* Same function as getRevision but returns the revision number as string or “merged” for a merged tree.

- *merge:* Add the given tree to the current tree.
- *removeRevision:* Remove the given revision number from the tree.
- *updateNodeIDs:* Gives all nodes in the tree a unique ID.

4.1.2 SolidSXPackageNode

A SolidSXPackageNode is a SolidSXNode with as type “package”. There are no additional public methods.

4.1.3 SolidSXClassNode

A SolidSXClassNOde is a SolidSXNode with as type “class”. There are no additional public methods.

4.1.4 SolidSXFieldNode

A SolidSXFieldNode is a SolidSXNode with as type “field” and the following additional public method:

- *getFieldType:* Returns a string with the type of the field.

4.1.5 SolidSXMethodNode

A SolidSXMethodNode is a SolidSXNode with as type “method”. There are no additional public methods.

4.1.6 SolidSXConstructorNode

A SolidSXContructorNode is a SolidSXMethodNode with as type “constructor”. There are no additional public methods.

4.2 SolidSXReference

The SolidSXReferences is used to store a reference path and type. The following types are available: Import, Call, Uses, Access, IsA and Inheritance. It includes the following public methods:

- *getPath:* Returns the reference path.
- *getType:* Returns the reference type.