Test Results Excel Report Details

Summary

The function of reporting the JUnit test results in the Excel format is not an open source. It was developed in the e- government development framework development environment. Here, let' s look at Excel Report function in details

Functionality

Automatized JUnit TestCase leaves results which can be checked in Test Reporting. In Korea, there are various reports and controls prepared in Excel format and therefore in addition to Text, XML and HTML provided by default, a function that can produce the Excel format is provided as Ant Task and Maven Plug- in.

- The test results can be provided by using Ant and Maven.
- The Excel template style can be customized for use.
- Basic template Excel file is provided.

For details on how to use, refer to Test Reporting and Test Results Excel Report.

Structure

The structure of a code is as follows.

Project

It is composed of three projects.

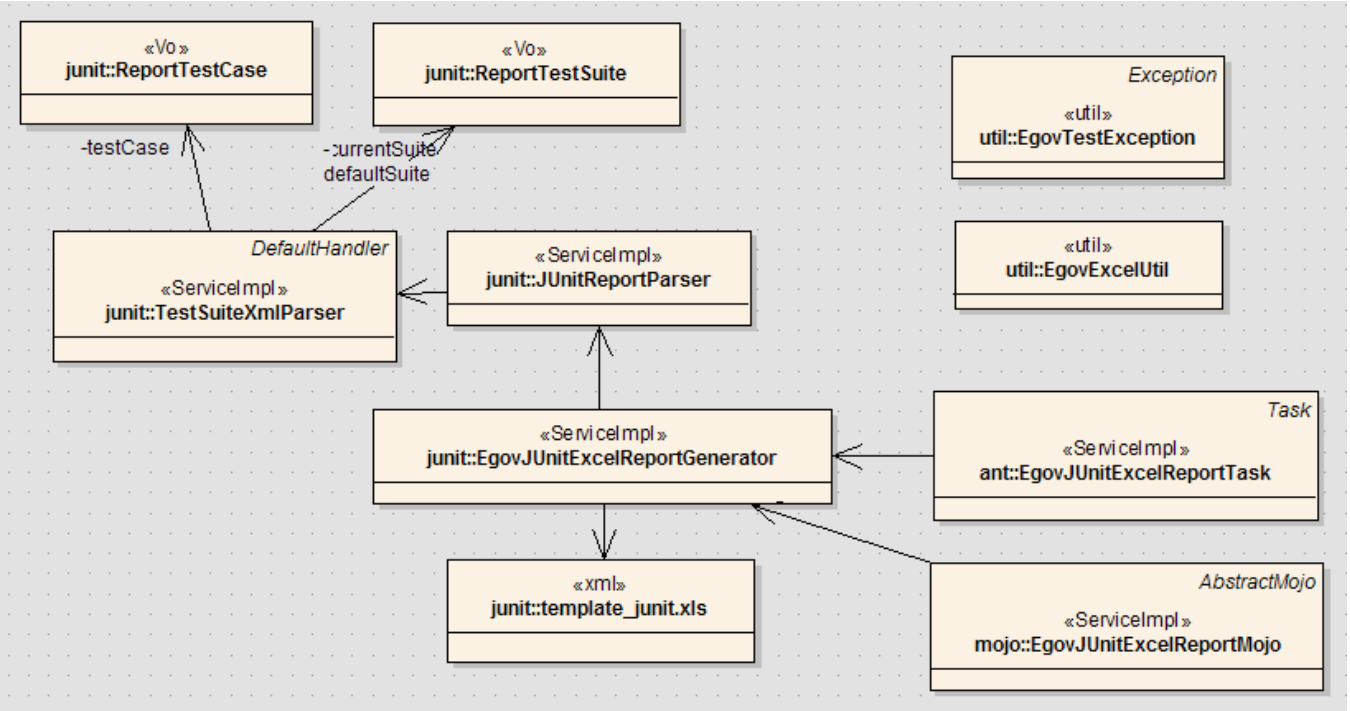
Project name	*Build tool*	*Description*
egovframework- dev- tst	Maven pom.xml	It carries out the core task of parsing JUnit Test XML Report file and reading the template Excel file and then mapping the values to produce an Excel report file
egovframework- dev- tst- ant	Ant build.xml	Use egovframework- dev- tst.jar file to provide Ant Task.
egovtest- maven- plugin	Maven pom.xml	Use egovframework- dev- tst.jar file to provide Maven Plug- in.

Package

Project name	*Package name*	*Description*
egovframework- dev- tst	egovframework.dev.tst.report.junit	This contains the class for core tasks of parsing JUnit Test XML Report file and reading the template Excel file and then mapping the values to produce an Excel report file
egovframework- dev- tst	egovframework.dev.tst.report.util	This contains Exception and Excel handling utility class
egovframework- dev- tst- ant	egovframework.dev.tst.report.ant	This contains EgovJUnitExcelReportTask class (Ant Task).
egovtest- maven- plugin	egovframework.dev.tst.mojo	This contains EgovJUnitExcelReportMojo (Maven Plugin).

Entire Class Structure

The organization of the entire classes are as follows.



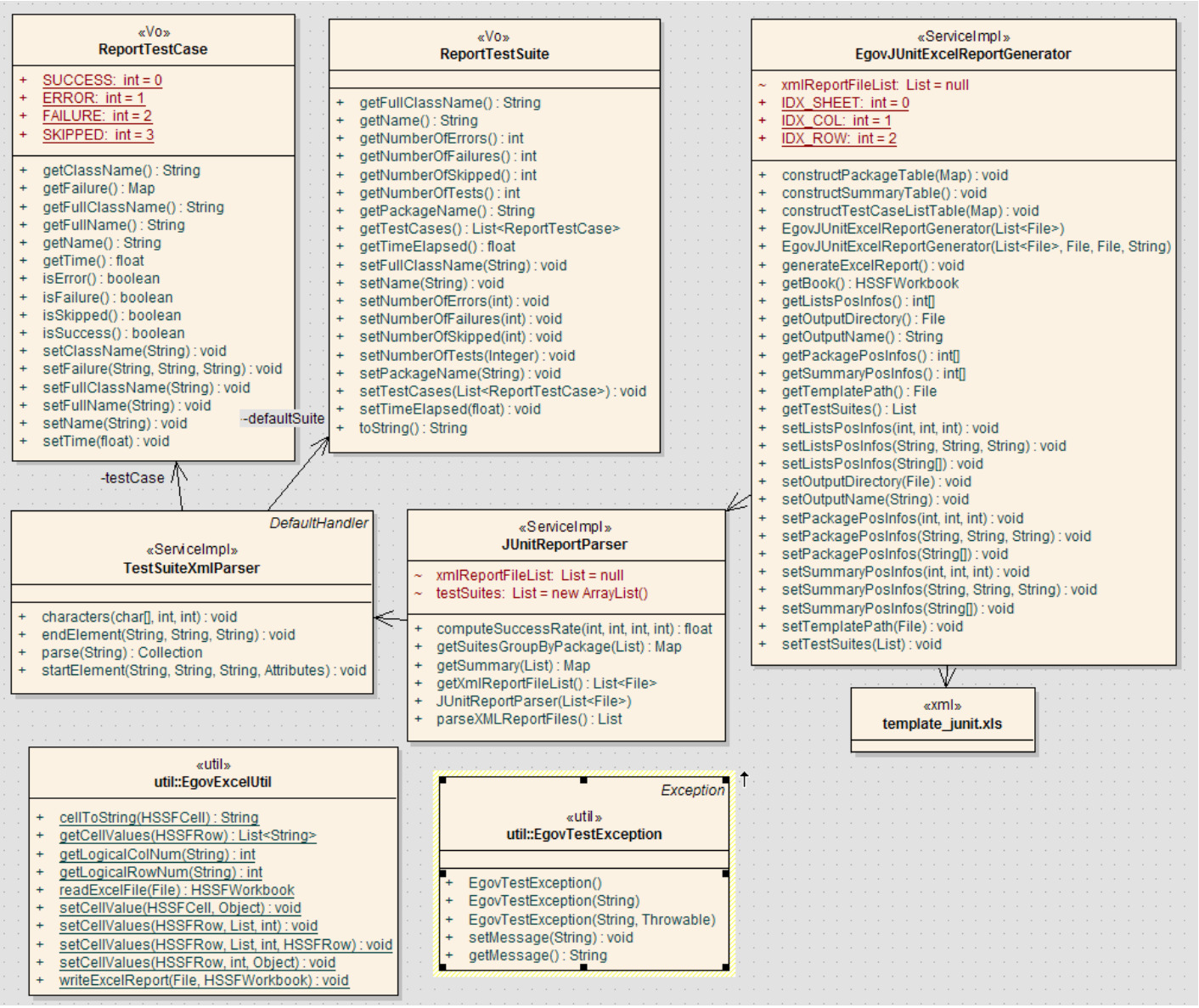
The relations are described as in the following.

Project name	***Class***	***Description***
egovframework- dev- tst	ReportTestCase	VO class that stores TestCase informaton, the smallest unit of test results.
egovframework- dev- tst	ReportTestSuite	VO class that stores TestSuite information, a collection of TestCase
egovframework- dev- tst	TestSuiteXmlParser	Inherit org.xml.sax.helpers.DefaultHandler and parse JUnit Test XML file, and put it in each VO class.
egovframework- dev- tst	JUnitReportParser	Use stored VO class to reorganize information for reporting (Summary, Package, TestCase Lists)
egovframework- dev- tst	EgovJUnitExcelReportGenerator	Read reporting information organized by JUnitReportParser from the template Excel file to organize the list and produce Excel report file.
egovframework- dev- tst	EgovExcelUtil	Utility class that uses Excel- related Apache POI in EgovJUnitExcelReportGenerator
egovframework- dev- tst	EgovTestException	Exception class.
egovframework- dev- tst	template- junit.xls	Default template Excel file for Excel report production
egovframework- dev- tst- ant	EgovJUnitExcelReportTask	Ant Task class implemented by inheriting org.apache.tools.ant.Task
egovframework- dev- tst- ant	egovtest.properties	Property file that defines Task name to be used as Ant Task
egovtest- maven- plugin	EgovJUnitExcelReportMojo	Maven Plugin Goal class implemented by inheriting org.apache.maven.plugin.AbstractMojo
egovtest- maven- plugin	egovtest- report.properties	Property file used in Maven Plug- in
egovtest- maven- plugin	META- INF/maven/lifecycle.xml	Setup file that defines the lifecycle of Maven

		Plug- in

egovframework- dev- tst project

The egovframework- dev- tst project includes JUnit Test Excel Generator function. The classes are described in the following.



✔ For description about `egovframework-dev-tst` project class, refer to the API documents and relevant sources. Find related information in egovframework- dev- tst project source code.

Structure of JUnit Test XML

Let' s discuss the structure of XML file produced after Junit test. XML file is produced in Maven and Ant, but it has the same format of TEST- ***[TestCase Full Name]***.xml. XML structure is as follows.

- ***testsuite*** : summarized information on test class.
  Attributes : ***errors, skipped, tests, time, failures, name***
  - properties : information on the environment where test class is tested.
    - property : value
  - ***testcase*** : information on test methods of test class.
    Attributes : ***classname, time, name***

- skipped : test method set up with @Ignore
- error : error information, if any.

EgovJUnitExcelReportGenerator

EgovJUnitExcelReportGenerator class reads the template excel file, map the test results and produce it in an Excel report. EgovExcelUtil is used to handle Excel files. This class and EgovExcelUtil class used Apache POI, an open source in order to handle Excel files. For details, refer to Apache POI Quick Guilde. In the document, EgovJUnitExcelReportGenerator class is described as standard.

Fields

Classification	*Input parameter*	*Description*	*Default*
Information to get test result information from XML	JUnitReportParser parser	JUnit XML Parser	N/A
Information to get test result information from XML	List<File> xmlReportFileList	JUnit Test XML file Full Path list	Treated as an error if N/A
Information to get test result information from XML	List<ReportTestSuite> testSuites	Test Suite Lists	N/A
Information required to produce Excel file	File templatePath	Full Path of template report file of Excel report	File object of " /template- junit.xls"
Information required to produce Excel file	File outputDirectory	Directory where Excel file is produced	Treated as an error if N/A
Information required to produce Excel file	String outputName	Excel report file name	" egovtest- junit.xls"
POI Excel file information	HSSFWorkbook book	POI excel file information	N/A
Header location information in template Excel file	int[] summaryPosInfos	Summary header location information	{ 0, 0, 3 }
Header location information in template Excel file	int[] packagePosInfos	Package header location information	{ 0, 0, 7 }
Header location information in template Excel file	int[] listsPosInfos	TestCase Lists header location information	{ 1, 0, 3 }
Constant	IDX_SHEET = 0 IDX_COL = 1 IDX_ROW = 2	Index constant of array that contains the header location information of template Excel file. sheet, column, row sequence	

Constructor

The following informaton should be inputted from the constructor.

- List<File> xmlReportFileList : JUnit Test XML file' s Full Path List
- File templatePath : template Excel file object
- File outputDirectory : file production location object
- String outputName : name of Excel file to be produced

Methods

- generateExcelReport() : main function that produces Excel files. Refer to EgovJUnitExcelReportGenerator]]
- private void init() : issues an error message if there is information not inputted.
- constructSummaryTable() : fills up Summary Table .

- constructPackageTable(Map suitePackages) : fills up the Table per package .
- constructTestCaseListTable(Map suitePackages) : fills up the table list per TestCase .

EgovExcelUtil – Excel file handling

For Excel file handling, Apache POI' s HSSF object is used. For detailed source code, refer to EgovExcelUtil.

- EgovExcelUtil.readExcelFile : read Excel files to produce Workbook.
- EgovExcelUtil.writeExcelFile : store in an Excel file.
- EgovExcelUtil.setCellValues : Set values in Cells.
- EgovExcelUtil.getLogicalRowNum : replace the actual row value (1- based) with logical 0- based row value.
- EgovExcelUtil.getLogicalColNum : replace the alphabets (actual column name, e.g, A to Z) with logical 0- based column values.
  ✔ The two- digit columns (eg, AA ~ ZZ) have not been implemented as there is little possiboiity of use, but if required, this logic should be implemented.

Excel File Handling Using POI

For example of using Apache POI to handle Excel files, refer to Apache POI Quick Guilde.

egovframework- dev- tst- ant Project

The EgovJUnitExcelReportTask class is as follows. For a brief source description, refer to egovframework- dev- tst- ant project source code.

The EgovJUnitExcelReportTask class is implemented to fit the Ant Task framework. Let's explore the structure from the viewpoint of Ant Task.

1. Define the structure to be used in ***build.xml***. In other words, define the function and attribute to be inputted. – refer to Excel Report Production Sample.
    - simple build.xml

      ```
      <egov-junitreport todir="${testxls.dir}">
          <fileset dir="${testreports.dir}" includes="**/TEST-*.xml" />
      </egov-junitreport>
      ```

    - full setting build.xml

      ```
      <egov-junitreport todir="${testxls.dir}"
                        outputname="egovtest-junit-full.xls"
                        templatepath="${basedir}/build/template-kr.xls"
                        summary="0,B,6"
                        packages="0,B,11"
                        lists="1,A,5">
          <fileset dir="${testreports.dir}"
                   includes="**/TEST-*.xml" />
      </egov-junitreport>
      ```

2. Produce Ant Task class that inherits org.apache.tools.ant..

   ```
   public class EgovJUnitExcelReportTask extends Task {
   ```

3. Declare the attribute to be inputted from build.xml as Field and produce the method for input. Declare Attribute as a variable and product a setter for automatic value setting.

   ```
   /** Directory where Excel files are produced */
   String todir;

   public void setTodir(String todir) {
       this.todir = todir;
   }
   . . . . . .
   ```

4. Declare FileSet to be produced after receiving from fileset tag and produce the method for input.

   ```
   List filesets = new ArrayList();

   public void addFileset(FileSet fileset) {
       filesets.add(fileset);
   }
   ```

5. Product functions to be handled in the public void execute() throws BuildException method.

egovtest- maven- plugin project

The EgovJUnitExcelReportMojo class is as follows.

The EgovJUnitExcelReportMojo class is a plug- in that provides by using the artifactID called ***egovtest*** and goal called ***junit- xls*** to provide Excel reporting function to Maven.
You can produce plug- ins that provide functions to Maven, if you have some tools. For details, refer to http://maven.apache.org/plugin- developers/index.html.

Write Maven Plugin Project

- Select New Maven Project and then ***GroupID:org.apache.maven.archetypes, ArtifactID: maven- archetype- mojo*** as archetype to create a basic structure.
- groupID and artifactID definitons : Name as in ***maven- ${prefix}- plugin or ${prefix}- maven- plugin*** format, in order to use prefix to shorten them when calling goals in the future.
- The pom.xml is defined as in the following.

```
<groupId>egovframework.dev</groupId>
<artifactId>egovtest-maven-plugin</artifactId>
<packaging>maven-plugin</packaging>
<version>1.0.0</version>
<name>egovtest-maven-plugin Maven Mojo</name>
<url>http://maven.apache.org</url>

. . . . . .

        <dependency>
                <groupId>org.apache.maven.shared</groupId>
                <artifactId>maven-plugin-testing-harness</artifactId>
                <scope>test</scope>
                <version>1.1</version>
```

```
        </dependency>
        <dependency>
                <groupId>org.apache.maven</groupId>
                <artifactId>maven-plugin-api</artifactId>
                <version>2.0</version>
        </dependency>
```

Write Mojo Class

Mojo stands for Maven Old Java Object, meaning the goal of Maven2. For details, refer to http://maven.apache.org/guides/introduction/introduction-to-plugins.html.]]]). To write Mojo, you have to define some in advance.

1. Define what to set up in pom.xml that uses this plugin. Define the default value, input value and names. – refer to Excel Report Production Sample

2. Goal, phase and lifecycle to be used when using Mojo: refer to Build Lifecycle.
   Mojo is defined as in the following. This section is defined as the class annotation.

   ```
   /**
    * @goal junit-xls
    * @execute phase="test" lifecycle="egovtest"
    */
   ```

3. Define the class name (put Mojo at the end.) and inherit `org.apache.maven.plugin.AbstractMojo`.
   If you inherit this upper class, you can set up in <build>...</build> of pom.xml.

   ```
   import org.apache.maven.plugin.AbstractMojo;

   public class EgovJUnitExcelReportMojo extends AbstractMojo {}
   ```

4. Define the parameter for input. You can use annotation to define the parameter name, default value and whether to have the required values. Therefore you don't have to write a separate setter.

   ```
       /**
        * Directory where Excel files are produced
        *
        * @parameter expression="${outputDirectory}"
        *            default-value="${project.build.directory}/egovtest"
        * @required
        */
       private File outputDirectory;

       /**
        * Excel report file name
        *
        * @parameter expression="${outputName}" default-value="egovtest-junit.xls"
        * @required
        */
       private String outputName;

       /**
        * JUnit Test results XML file location for Excel report proeuction
        *
        * @parameter expression="${reportsDirectory}"
        *            default-value="${project.build.directory}/surefire-reports"
        */
       // private File[] reportsDirectories;
       private File reportsDirectory;

       /**
        * Header location information of template report file of Excel report
   ```

```
     *
     * @parameter expression="${headerPosition}"
     */
    private Map headerPosition;

    /**
     * Full Path of template report file of Excel report
     *
     * @parameter expression="${templatePath}"
     */
    private File templatePath;

    /**
     * Maven Project
     *
     * @parameter expression="${project}"
     * @required @readonly
     */
    private MavenProject project;
```

5. Write main functions in the public void execute() method.

```
    public void execute() throws MojoExecutionException, MojoFailureException {
...
    }
```

Expansion Strategies

Main functional Expansion

1. Add the class that writes main functions to the egovframework- dev- tst project.
2. For package name, write sub package under report as in egovframework.dev.tst.report.emma.
3. Like the classes of egovframework.dev.tst.report.junit package, write VO class, XMLParser, ReportParser, ExcelReportGenerator and other classes to suit their functions.
4. Use Maven deploy to deploy up to the Nexus sever.

Add Ant Task

After creating main functions, write Ant Task if required.

1. Add Ant Task class to the egovframework- dev- tst- ant project.
2. Under egovframework.dev.tst.report.ant package, write the class name like EgovJUnitExcelReportTask.
3. Refer to egovframework- dev- tst- ant project to write Ant Task.

Add Maven Plugin

After creating main functions, write Maven Plugin if required.

1. Add Mojo class to the egovtest- maven- plugin project.
2. Under egovframework.dev.tst.mojo package, write the class name like EgovJUnitExcelReportMojo.
3. Refer to Write Mojo class production to write Mojo class. At this time, just redefine the goal only. The rest is the same with the existing Mojo.

TestCase Decription

This function wanted three projects to be implemented in TDD(Test- driven Development) type. Therefore regardless of whether written before or after implementation, TestCases were written for each class and the test code was written with a focus on the functions that each class handles.

egovframework- dev- tst TestCase

The TestCases of egovframework- dev- tst project are general one that mainly uses JUnit4 and Unitils. The data used by each test code is located in the resources directory. Especially for the `JUnitExcelReportGeneratorTest` class, a test is included that is to see if Excel files are properly produced by using Apache POI and style and header values are properly written.

egovframework- dev- tst- ant TestCase

The TestCase of egovframework- dev- tst- ant package is a test code to test Ant Task. Ant provide a test framework which is an extension of JUnit 3.8 for Task test. Through this, you can check if what you set up in the build.xml works properly.

1. Write build.xml for testing. – The writing is the same with general build.xml file. Make a direct reference to the source.
2. `Inerit the org.apache.tools.ant.BuildFileTest` to write TestCase.

   ```
   import org.apache.tools.ant.BuildFileTest;

   public class JUnitExcelReportTaskTest extends BuildFileTest
   ```
3. Read the build.xml for testing to organize a virtual project.

   ```
   public void setUp() throws Exception {
       configureProject(TestUtil.getBaseDirString(this.getClass()) +
   "/unit/egovtest-excel-test/build.xml");
   }
   ```
4. `Use various meothds defined in the org.apache.tools.ant.BuildFileTest to write a test code.`

egovtest- maven- plugin TestCase

Even writing Mojo, provide a base on which the pom.xml can be tested.

1. Write the pom.xml for testing.
2. `Inherit the org.apache.maven.plugin.testing.AbstractMojoTestCase` class to write TestCase.

   ```
   import org.apache.maven.plugin.testing.AbstractMojoTestCase;

   public class JUnitExcelReportMojoTest extends AbstractMojoTestCase {}
   ```
3. Read the pom.xml for testing.

References

- Apache POI : http://poi.apache.org/
  - Busy Developers' Guide to HSSF and XSSF Features : http://poi.apache.org/spreadsheet/quick- guide.html
- Write Custom Ant Task
  - Apache Ant Manual : http://ant.apache.org/manual/
  - Introduction to Custom Ant Tasks : http://www.developer.com/java/article.php/3630721
  - More on Custom Ant Tasks : http://www.developer.com/java/article.php/3636196
- Maven Home : http://maven.apache.org/

    - Write Maven Plugins : http://maven.apache.org/plugin- developers/index.html

- Test Result Excel Report