D:\OE\Website\Design Commission Deliverables\Word Separated Logo\Object Edge - Final Logos\png\object-edge-logo-mini.png

Customer Logo or Name



Regression Testing Tool

User Guide

**Object Edge Inc.**

**315 Lennon Lane**

**Walnut Creek, CA-94598**

Document History

| Date | Version | Author | Revision |
| --- | --- | --- | --- |
| 2013/01/10 | 0.1 | Igor Abramovich | Initial Version |
| 2013/02/13 | 0.2 | Igor Abramovich | Enhancements and bug fixes (see Release Notes) |
| 2013/02/15 | 0.3 | Igor Abramovich | Usability enhancements (see Release Notes) |
| 2013/02/21 | 0.4 | Kiran Gawde | Logging, run-time diagnostics enhancements |
| 2013/02/27 | 0.5 | Igor Abramovich | Misc JIRA enhancements (see Release Notes) |
| 2013/07/09 | 0.11 | Uma Parakala | Usability enhancements (see Release Notes) |
| 2013/07/31 | 0.11 | Uma Parakala | Usability enhancements (see Release Notes) |
| 2014/01/20 | 0.11 | Uma Parakala | Changes to some features |

Table 1‑1: Document History

Preface

Scope

Purpose

Audience

The document is primarily targeted at the following groups:

Acknowledgements

We gratefully acknowledge the training received from and contributions from …… teams. We are grateful to ………. for all their assistance and guidance. Finally, we are especially grateful to <big bosses> for providing us this opportunity and for placing their confidence in our services.

Table of Contents

[1 Executive Summary 5](#_Toc363035711)

[2 Logical Model 6](#_Toc363035712)

[2.1 SiteSpec 6](#_Toc363035713)

[2.2 Test Data Suites 6](#_Toc363035714)

[2.3 Test Data Flows 6](#_Toc363035715)

[2.4 (Custom) Action Library 7](#_Toc363035716)

[2.5 Data and Flow templates 7](#_Toc363035717)

[2.6 Diagram 7](#_Toc363035718)

[3 Using the Tool 9](#_Toc363035719)

[3.1 Structure of a Test Project 9](#_Toc363035720)

[3.2 Typical Workflow 10](#_Toc363035721)

[3.3 Development tasks 11](#_Toc363035722)

[3.3.1 Creating/bootstrapping a new testing project 11](#_Toc363035723)

[3.3.2 Creating/Maintaining SiteSpec 11](#_Toc363035724)

[3.3.3 (Re)Generating data and flow templates 11](#_Toc363035725)

[3.3.4 Managing test data flows 12](#_Toc363035726)

[3.3.5 Managing test data 13](#_Toc363035727)

[Test Data ID 13](#_Toc363035728)

[The system expects unique test data IDs across data files in a project. For this reason, the filenames of input data files need to be unique within a project (even if their folder paths are different). 13](#_Toc363035729)

[Compound Test Data 13](#_Toc363035730)

[Variables 14](#_Toc363035731)

[3.3.6 Custom actions (for Developers) 14](#_Toc363035732)

[3.3.7 Updating Reference Data 15](#_Toc363035733)

[4 Run-time tasks 16](#_Toc363035734)

[4.1.1 Setting run-time properties 16](#_Toc363035735)

[4.1.2 Setting Browser Properties 16](#_Toc363035736)

[4.1.3 VM arguments 16](#_Toc363035737)

[4.1.4 Setting the Web URL environment 17](#_Toc363035738)

[4.1.5 Exit program on a flow execution Error 17](#_Toc363035739)

[4.1.6 Run to and from a certain transition 17](#_Toc363035740)

[4.1.7 Run a range of test cases 17](#_Toc363035741)

[4.1.8 Example Execution 18](#_Toc363035742)

[4.1.9 Running flows 18](#_Toc363035743)

[4.1.10 Examining execution results 18](#_Toc363035744)

[5 Using Results Aggregator 19](#_Toc363035745)

[5.1.1 Setting up the program 19](#_Toc363035746)

# Executive Summary

Site Regression Testing Tool is designed to enable Analysts and Testers develop, maintain and execute regression tests of arbitrary websites. With very few exceptions, data required to model the tests is externalized, as Excel, Json, XML or property files. This user guide has two parts: high-level logical model and concrete set of tasks involved in using the tool. The tool, and consequently the user’s guide, is intended for Testers and Business Analysts. There are only couple Developer tasks, which we will point out.

# Logical Model

The tool supports two modes of usage: development and run-time. During development, Testers, with assistance from Business Analysts and Developers, develop and maintain externalized artifacts, representing their tests. These artifacts are described below. There will be an example of how to create and use these artifacts in the second part of the document. At run-time, testers execute set of data flows and examine generated results.

## SiteSpec

XML-based ‘map’ of the website being tested, which includes all the **elements** of the site, arranged in a hierarchy: site->page->section->element. Elements refer to text fields, buttons, drop-downs, etc. Each element has a unique name, which can be used to refer to it and to associate a data element with it.

## Test Data Suites

A **test-data** is a simply a set of property-value pairs, where properties are siteSpec element names. In addition, each test-data has a unique id. For example, <id: “userCredentials -1”, userName: “igor”, userLogin: “oeigor”>. Typically, quality regression tests require multiple ‘instances’ of test-data, to support different conditions. For example, <id: “userCredentials-2”, userName: “kiran”, userLogin: “oekiran”> would be required to support tests for a different users. In addition to primitive elements, test-data can be compound, or refer to other test-data instances, via their ids. For example, a test data can refer to oeigor via <userCredentials: userCredentials-1”.

A **test-data-suite** is a collection of test-data, which share their properties. A test data suite also has a unique id and a set of test-data. By convention, test data ids within a suite have the syntax: <testSuiteId-1>, <testSuiteId-2>, etc. So the two test-data instances above could belong to the suite “userCredentials”.

## Test Data Flows

A test data flow is a sequence of steps (or **transitions**), navigating through a website, using test data. Each transition has **transition-id, transition action, input data, destination page, and expected data**. Some transition actions are ‘out-of-the-box’, based on the standard set of Selenium[[1]](#footnote-1) APIs (MoveToElement, Fill Data, Select an Element, Push a Button, etc). Transition actions can only be custom-built (see below).

## (Custom) Action Library

Each transition of test data flow includes an action which defines type of the transition. In addition to out-of-the-box set, test applications (or group of applications) can provide their own custom actions. For example, *Logout* action would look for a logout button and select it. Custom actions are Java classes, which subclass *UserDefinedTestDataAction* and implement *TransitionTestDataAction* interface.

Custom actions are created by developers, per specification of a Business Analyst.

## Data and Flow templates

Both data suites and flows are defined using Excel. In order to simplify development, corresponding templates are generated from SiteSpec. Templates contain meta-data, extracted from the SiteSpec (page and element names) as well as custom actions, found on the libpath. Templates are typically generated once, to bootstrap a project.

## Diagram

The diagram below shows the relationship among model artifacts, in development and run-time modes. On the left there are user-actions, exposed to both testers, via commands, and to developers, via Java APIs:

* generateDataTemplate: generate a data template based on the SiteSpec
* generateFlowTemplate: generate a data flow template based on the SiteSpec
* updateReferenceData: update reference information, based on modified SiteSpec, in all test data and flow artifacts.
* runFlows: run a set of flows, either all application flows or a subset based on a filter.



# Using the Tool

## Structure of a Test Project

A test project is an Eclipse project, which contains all application-specific test artifacts. Non-Java artifacts live in sub-folders of the top-level **resources** folder. The only Java classes in a test project should be custom actions, if there are any.

The diagram below maps logical artifacts to their implementation equivalents.



## Typical Workflow

The diagram below shows a typical workflow to create and maintain a regression test project.

Below we describe the workflow tasks in more detail.



## Development tasks

### Creating/bootstrapping a new testing project

1. Create an Eclipse project with oe\_regression\_testing\_tool as a prereq.
2. Load oe\_regression\_testing\_tool\_sandbox project – it contains some useful examples and templates. There is no need to prereq this project.
3. Copy *resources* folder from \oe\_regression\_testing\_tool to your project. There will be the following subfolders:
   * data (empty)
   * flows (empty)
   * properties (3 property files)
   * results (empty)
   * runConfigs (5 files)
   * templates (1 internal file)

### Creating/Maintaining SiteSpec

Create a Json file <appSiteSpec>.json directly under resources directory. The file should contain all the elements referenced in tests, in the context of their parent sections and pages. You can use oe\_regression\_testing\_tool\_sandbox/resources/aaSiteSpec.json as an example. Elements can be identified (located) by id, name or xpath, exactly as in Selenium. Refer to Selenium documentation for details on element locating: <http://seleniumhq.org/docs/02_selenium_ide.jsp#locating-elements>.

Once you decide on the name of the siteSpec file, update following files to point to your siteSpec

1. *resources\properties\TestDataManager.properties*
2. *resources\properties\TestFlowManager.properties*

### (Re)Generating data and flow templates

After you have completed the siteSpec, invoke *GEN\_DATA\_TEMPLATES* and *GEN\_FLOW\_TEMPLATE* runconfigs. *GEN\_DATA\_TEMPLATES*  will generate a directory for each page, containing the page template and templates for each page’s sections. GEN\_FLOW\_TEMPLATE will generate resources/templates/TestFlow.xls.

If you make any changes to the SiteSpec, you can always regenerate the templates.

Examine the templates. In the data templates you should see list of all element names in the element drop-downs in the first row, restricted by the template’s page or a section. In the data flow template there are three flow transitions, one for each type of input data: CONTROL, INPUT\_DATA and INPUT\_DATA\_SUITE. As you build individual flows, you can copy-and-paste transitions of desired types from this template. These three types are described below.

### Managing test data flows

Copy *resources*/*templates/TestFlow.xls* into *resources/flows* directory, rename it and use it to create your first data flow.

A data flow has set of properties and a list of transitions.

A transition can refer to an instance of *TestData* or to a *TestSuite*. It may also not refer to any application data, but to a page element (for example button push). These three types of transitions are supported by corresponding input data types: *TEST\_DATA, TEST\_DATA\_SUITE and CONTROL*.

*TEST\_DATA\_SUITE transitions* are special because they produce multiple flow instances when bound to each test data of the transition’s test suite. Before a flow is executed, the framework resolves all test suite references and determines number of flow instances to be the smallest of the referenced test suites. For example, if a transition-1 references a testSuite of size 2 (i.e. it has 2 test data rows) and transition-2 references a testSuite of size 3, enclosing flow will be executed 2 times. On each iteration, suite transition will operate on the test data under cursor, for corresponding test suite. Ideally, test data suites referenced by a flow should be of the same size.

*TEST\_DATA transitions* will execute on the same instance of test data on each iteration. This can be useful if you want to keep some element constant while varying others (e.g. constant userLogin per flow).

CONTROL transitions do not refer to any test data. Instead they may optionally refer to an element. For example, pushButton action requires button element name as an argument. Elements, available, in the drop-down are restricted to the destination page of the previous transition (or flow start page for the first transition).

There is another input data type: NONE. It is only used when calling a custom action which does not require any input data. For example, Logout action may have all the data it needs encapsulated within it.

*Flow properties* are specified, in properties sheet, as follows:

1. *startPageName*: pageName where the flow will start. The name can be selected from the drop-down with all pages names, defined in the SiteSpec.
2. *SiteSpec*: hyperlink to the SiteSpec; should be pre-set in the template
3. *id*: flow id, must be set (once you set the id, rename the *flowTemplate* sheet to have the id as its name)

*Transition properties* are specified, for each transition, in the main sheet, as follows:

1. *Transition ID*: transition id, by default transition-1, transition-2, etc. The id should be descriptive, describing the transition.
2. *Transition Action*: an action, selected from column drop-down. The list should be pre-populated with both out-of-the-box, framework, actions and custom actions, defined in your application.
3. *Input Data Type*: One of the 3 supported types, CONTROL, TEST\_DATA or TEST\_DATA\_SUITE, or NONE.
4. *Input Data*: reference to an element, test data or test data suite, depending on the input data type. For element references, select desired element from dropbox. For TestData references, use testDataId of the desired test data instance. For TestDataSuite, use testDataSuite id. For the latter two cases, use hyperlink with address pointing to the test suite file (hyperlink is used only for Excel navigation. At run-time, only cell value is used to identify the test data). For fill and select actions, input data suite can be a page-level or section-level data. Section-level data can spawn multiple sections. System will match values with elements based on element (or column) names. Therefore, for section-level data, element names must be unique across sections.
5. *Destination Page*: Destination page of the transition, selected from the list of siteSpec pages.
6. *Expected Data Suite (optional)*: data suite which represents partial state of the target page. The current data will be validated against expected, based on correlated element names. For each expected element name, corresponding element will be found on the current page and its values extracted for comparison.

### Managing test data

Copy *resources/templates/TestData.xls* into *resources/data* directory and use it to create your first data file. Test data is simple a collection of property-value pairs. The tricky cases involve *compound test data* and *variables*.

#### Test Data ID

The test data ID has to be composed of the following three parts in the order listed below.

1. Name of the test data file without the file extension
2. A hyphen
3. An integer

Example: File name=”SearchData.xls, the test data ID could be “SearchData-1”, “SearchData-2”, “SearchData-501” etc.

The system expects unique test data IDs across data files in a project. For this reason, the filenames of input data files need to be unique within a project (even if their folder paths are different).

#### Compound Test Data

Any test data can refer to another instance of test data via test data id. When entering such value, create a hyperlink with target testDataId as textToDisplay target test data suite as address. This is usually done when a page has multiple sections, each represented as a test data. For example, a sign up page may have two sections, *signin* and *registration*. To represent two test data instances, SignUpPageData test suite will have two test data rows, with properties for section element names:

|  |  |  |
| --- | --- | --- |
| TEST\_DATA\_ID | Signin | Registration |
| **signUpPageData-1** | [signUpSection1-1](file:///C:\eclipse-workspaces\qaFramework\oe_regression_testing_tool_sandbox\resources\data\registration\signUpSection1.xls) | [signUpSection2-2](file:///C:\eclipse-workspaces\qaFramework\oe_regression_testing_tool_sandbox\resources\data\registration\signUpSection2.xls) |
| **signUpPageData-2** | [signUpSection1-3](file:///C:\eclipse-workspaces\qaFramework\oe_regression_testing_tool_sandbox\resources\data\registration\signUpSection1.xls) | [signUpSection2-4](file:///C:\eclipse-workspaces\qaFramework\oe_regression_testing_tool_sandbox\resources\data\registration\signUpSection2.xls) |
|  |  |  |

signUpSection2, in turn, must also have two test data rows, with properties for leaf element names:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TEST\_DATA\_ID | RegistrationPassword | signUpConfirmPassword | firstName | lastName |
| **signUpSection2-2** | 123abcd | 123abcd | Kiran | Gawde |
| **signUpSection2-4** | 123456a | 123456a | Igor | Abramovich |

A transition referring to SignUpPageData test suite will be executed twice, once for each test data.

#### Variables

Sometimes there is a requirement where data must be dynamic, for example, a unique userId to test a login, or today’s date. The system supports this by allowing Java format variables within test data values. For example, signUpSection1 data suite, which requires unique set user ids, can be specified as follows:

|  |  |
| --- | --- |
| TEST\_DATA\_ID | registrationUserID |
| **signUpSection1-1** | igor{0,number,#}@ objectedge.com |
| **signUpSection1-3** | kiran{1,number,#}@objectedge.com |

At run-time, variables will be bound to randomly-generated values (e.g. [igor2345@objectedge.com](mailto:igor2345@objectedge.com), [kiran3245@objectedge.com](mailto:kiran3245@objectedge.com)). The scope of these binding will be the flow’s instance. Which means, if a variable is reused with a flow, it will be bound to the same value. This is particularly convenient for specifying expected values.

### Custom actions (for Developers)

Custom actions allow higher-level transition types and are implemented as Java subclasses com.objectedge.site.test.model.execution.UserDefinedTestDataAction. There are two required methods to implement: getName and execute. getName returns a String, which will appear in the set of allowed actions in flow editor; execute methods defines the behavior. Typically custom actions are implemented in terms of Selenium API. There are few examples of custom actions in the *oe\_regression\_testing\_tool\_sandbox* project.

If you define custom actions, you may need to modify *actionClassPackages* in properties/TestFlowManager.properties to include the corresponding Java package. By default, the current values contain the framework package but you can add your own:

*actionClassPackages=com.objectedge.site.test, com.your.package*

### Updating Reference Data

If you make any changes to SiteSpec or custom actions, you can update corresponding meta-data in all test data and flows, by running UPDATE\_REF\_DATA runconfig.

This update will also trigger referential integrity validation, which will report any invalid references to data, page names, actions and elements.

NOTE: Make sure all data and flow XLS files are closed when you run UPDATE\_REF\_DATA. Otherwise system will not be able to update the files.

### Quick Guide to Transition Actions

# Run-time tasks

### Setting run-time properties

Run-time properties are defined in *resources/properties/TestFlowExecution.properties file.* The following properties are supported:

* newBrowserPerTest: If true, open a new browser for each flow, false – otherwise.
* loggingLevel: lowest log4J logging level (INFO, DEBUG, ERROR, etc)
* loggingTargets: targets where logs are written. Allowed values are CONSOLE and/or FILE, which correspond to System.out and a timestamped log file resources/logs/<{todayDate}/{timeNow}.log. Empty value will turn execution logging off.
* closeBrowserAtTheEndOfFinalTest: if true, close the browser at the end of the final test.
* timeoutElementWait: maximum wait, in seconds, before elements are presented. When looking for page elements, if an element is not found, system will assume it is because it has not been loaded yet. It will wait until it is, subject to this timeout.
* timeoutPageLoad: page loading timeout, in seconds.

### Setting Browser Properties

Browser properties are defined in renner\_test/resources/properties/Browser.properties file.

* browserType : defines the type of browser (IE, CHROME, FIREFOX)
* PATH\_CHROME\_DRIVER\_EXE : the pathname to the chrome web driver
* PATH\_IE\_DRIVER\_EXE : the pathname to the IE web driver

\*you can specify 32/64 bit driver versions. Check

“ ../oe\_regression\_testing\_tool/resources/browserDrivers” to located installed drivers.

We can quickly set the browser type in the arguments tab of run configurations.

Run Configurations -> arguments tab -> VM arguments

* Add VM arguments in this format, for example: -Dbrowser= FIREFOX

### VM arguments

These VM arguments have been added to the tool.

They can be set under: Run Configurations -> arguments tab -> VM arguments

1. “-Dbrowser” – specifies which browser to run program in
2. -DexitOnFirstError – debugging flag exits the program on first failed test case
3. -DfromTransition, DtoTransition, -DfromTestCase, -DtoTestCase - from/to testData & transition
4. -Denv– specifies the environment

### Setting the Web URL environment

The environment you specify for your VM argument “-Denv” will map to its corresponding website. This mapping is established in the SiteSpecGenerator.properties. For example, If you have environments named “QA2” and “OEQA” you would declare this is your SiteSpecGenerator.properties :

QA2=http://ecommerceqa2.lojasrenner.com.br/

OEQA=http://ecommercestage.lojasrenner.com.br/

NOTE: the environment key must be equal to the prefix associated to the different data files. For example, if we have data folder data\_qa2, we must have qa2= http://ecommerceqa2.lojasrenner.com.br/

If the incorrect VM environment name is given, the tool will set it to the default site in SiteSepc.xls and will store the results file in “Default” directory.

### Exit program on a flow execution Error

We have the option to exit the program when it encounters a failed execution during a flow. The failed test case will not be recorded in the final results file. This option is available for debugging purposes.

Run Configurations -> arguments tab -> VM arguments

* Add VM arguments in this format, for example: -DexitOnFirstError=Y (Y=set flag, N=disable flag)

### Specifying the environment directory for results

The VM argument “env” can be used to redirect your results files to that directory. If the directory does not exist, the tool will make a new directory. We can specify the arguments under

Run Configurations -> arguments tab -> VM arguments

* Add VM argument in this format, for example: -Denv=QA2

### Run to and from a certain transition

For debugging purposes, the user can run the flow from a certain transition to a certain transition. This can only be done under RUN\_ONE\_FLOW command.

To use this feature we would pass in the VM argument, for example, -DfromTransition=3

-DtoTransition=7. This will run transitions 3,4,5,6,7.

Use default values of -DfromTransition=1, -DtoTransition =-1 to run all the flow transitions.

Run Configurations -> arguments tab -> VM arguments

* Add VM arguments in this format, for example: - DfromTransition=4 -DtoTransition = 8
* Use default values of -DfromTransition=1, -DtoTransition =-1 to run all the flow transitions.

### Run a range of test cases

For debugging purposes the user can run a range of test cases. This can only be done under RUN\_ONE\_FLOW command.

To use this feature we would pass in the VM argument, for example, -DfromTestCase=3 -DtoTestCase=7. This will run test cases 3,4,5,6,7.

If we want to run only one test case, for example test case 5, we would pass in VM arguments like this: -DfromTestCase=5 -DtoTestCase=5

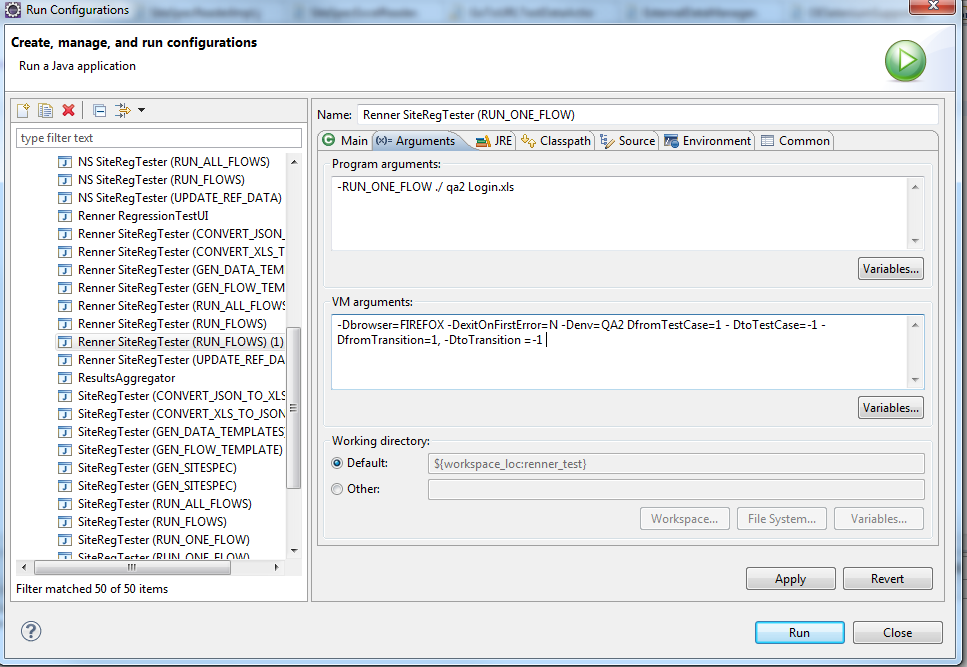
If you want to run all the test cases, use the default values -DfromTestCase=1 - DtoTestCase=-1

Run Configurations -> arguments tab -> VM arguments

* Add VM argument in this format for example: -DfromTestCase=1 -DtoTestCase=5

### Example Execution

This particular example runs only one flow. It uses the RUN\_ONE\_FLOW command. It runs the flow files Login.xls in the environment OEQA. It runs test case 1 with all the transitions.



### Running flows

There are two runconfigs to run flows: RUN\_ALL\_FLOWS and RUN\_FLOWS. The first one will run all flows in resources/flows directory. The second one takes a list of arguments, flow ids or wildcard specs. Flows whose ids are listed or whose ids match the wildcard spec will be executed.

### Examining execution results

Once a flow is executed, results are written to *resources/results/<environment>/<todayDate>* directory. For each run, a *<time>.CSV* file will be created. In the file, for each execution instance a row will be created.

For successful execution, row values are: success, flow-id, testDataIndex (within testDataSuite), startTimestamp, endTimestamp.

For failed execution, the row values are: failure, flow-id, testDataIndex (within testDataSuite), startTimestamp, endTimestamp, failedTransitionId, stackTrace.

# Using Results Aggregator

The Results Aggregator combines multiple executions of the same test flow and shows the most recent status of the flow execution. It merges the results based on its flowID, and browser type. It only displays the status of the latest flow execution.

The results aggregator program will work on the “results” directory in your project test file. Your results folder is organized into environment subfolders. The environment subfolder is further organized by dates(YYY-MM-DD). Within a specific date, we have the executed test flows named with timestamps

## Running the Results Aggregator

To run the results Aggregator we need to specify two VM arguments.

-Denv - Specify the environment folder.

-DdateDirectory - the date folder where the result files will be merged. If blank, it will refer to today’s date by default.

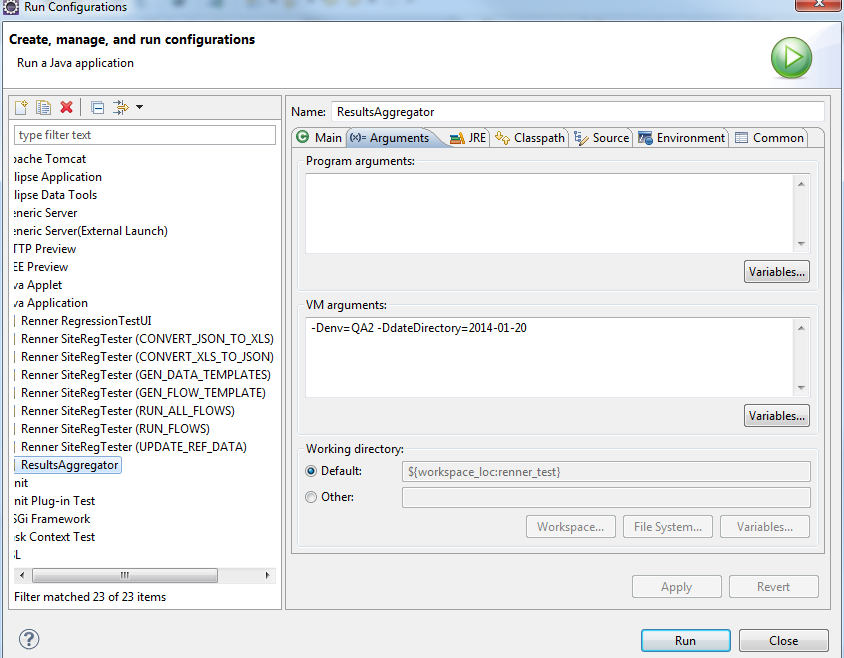
The merged file will be named merge along with the time stamp of creation. The rest of the files will be moved to an “Archives” folder.

Figure 1 merges result file in environment QA2 folder, under date folder 2014-01-20

1. Selenium is the engine used by the tool to run/execute test data flows. There is, however, there is a level of abstraction exists between the tool and Selenium so, in principle, Selenium can be easily replaced by a different engine with similar capabilities. Selenium documentation ia available at <http://seleniumhq.org/docs/03_webdriver.jsp>. [↑](#footnote-ref-1)