

Chapter 6

JSystem Automation Framework Functionality



➤ In this chapter...


<i>JRunner Functionality Overview</i>	<i>Page 2</i>
<i>Scenario Studio</i>	<i>Page 2</i>
<i>Working with Fixtures</i>	<i>Page 20</i>
<i>Tabbed SUT Editor</i>	<i>Page 22</i>
<i>JRunner</i>	<i>Page 25</i>
<i>General Operations</i>	<i>Page 39</i>


6.1 JRunner Functionality Overview

Once the developer has written a test or tests in java using the Eclipse development environment they are then saved in the class folder. The user then opens the JRunner and starts writing JSystem scenarios and executing them by opening the JRunner and activating the “**run**” batch script.

6.1.1 JRunner Environment Overview

The JRunner work environment is divided into two main modules, the scenario studio where the scenarios are built and configured and the scenario runner, where configured test scenarios are run, analyzed and published.

The scenario editor is divided into five tabs, the most important of which is the “**Tree Tab**” that display the list of tests as they were written in java code adhering to the java hierarchical layout convention. These tests are divided into classes and function methods that appear with check boxes alongside them. To create a new scenario the user simply checks the required method or methods, clicks on the  “**Add to Test Scenario**” button and the methods are added to the scenario runner pane.

The user can now run the test scenario by pressing the  “**play button**”, JSystem then signals the SUT and performs the test method functions that have been imported into the scenario runner from the scenario editor.

6.2 Scenario Studio

JSystem scenario studio is an Ant (<http://ant.apache.org/>) script file.

When creating a new scenario, an initial ant script file is created, when performing an operation on the scenario (for example adding a test), the ant script is updated.

6.2.1 Managing the Scenarios Repository

The following table lists the operations for managing the scenarios repository.






Name	Description	Button
New Scenario	Creates a new scenario.	
Copy Scenario	Saves a scenario using a different name.	
Save and Load Failed Sequence	Enables the user to create a scenario from all the tests that failed or were not activated in the last execution.	
Select Scenario	Selects a scenario to edit and or execute.	
Clear/Delete Scenario	Deletes current scenario and selects the “ default ” scenario.	

Table 1: Scenario Repository Operations

6.2.2 Editing a Scenario

The following table lists the operations for editing a scenario.









Name	Description	Button
Add item	Adds a test/scenario/fixture to the currently active scenario.	
Move Up Item	Moves test up in scenario order.	
Move Down Item	Moves test down in scenario order.	
Remove Item from Scenario	Deletes test/fixture/sub scenario from scenario.	
Edit with spreadsheet editor	Used to convert the scenario into an excel spread sheet format and then edit it and then save it automatically returning to the JSystem interface.	
Add Change SUT Event	This button enables the user to add and change the SUT event during the test run, changing the SUT file.	
Add Publish Results Event	Adds an event that automatically publishes the test results to date, after the test cycle reaches it in the already running scenario.	
View Test Code	Opens test java code in an HTML browser.	

Table 2: Editing Scenario Operations

6.2.3 Viewing and Editing Test Parameters

In order to edit and or view test parameter values in a scenario, either “**double click**” on a selected test in the scenario tree or just select the test and then select the “**Test Info**” tab.

The “**Test Info**” tab has three parts:

1. Full class name and method name.
2. Associated fixture and fail to fixture.
3. Execution time of last test activation.



Figure 1: Viewing and Editing Parameters

The snapshot above shows the “**Test Info**” tab with the general information section marked with red.

Test documentation, which contains two sub-tabs:

1. **Test documentation** - Tests the “**javadoc**” taken from test source code.
2. **User Test Documentation** – User input and comments text box. The JRunner user can add additional saved scenario information.

The snapshot shows the “**Test Info**” tab with the text documentation area.

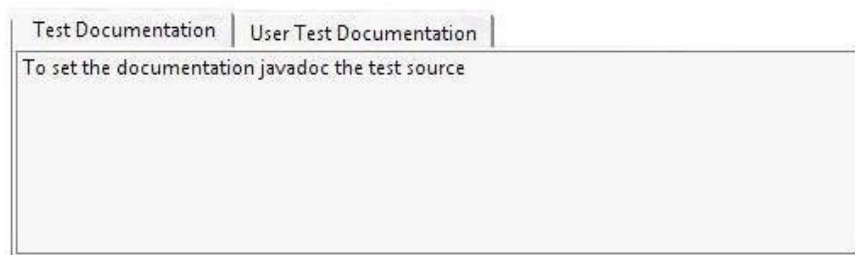


Figure 2: Test Info Tab

Test Parameters - Exposing test parameters in the “**Test Info**” tab is performed by the java programmer that authors the test.

Reference: *In order to understand the interaction between the test class and the parameters panel read Working with Parameter Values*
- Reference Example *on page 7.*

Note: *In order to edit test parameter values click on the value of the parameter and edit parameter value.*

The snapshot shows the “**Test Info**” tab with the text parameters.

General			
Name	Description	Type	Value
PingDestination		string	127.0.0.1
PacketSize		int	1024

Figure 3: Test Info Tab General

6.2.3.1 Sorting Parameters

In order to sort parameters “**click**” on the parameters table head. When clicking on the “**Name**” column head for the first time, the parameters are sorted by their name in an alpha numeric descending order, when clicking the second time, parameters are sorted by their name in an alpha numeric ascending order, when clicking for the third time, the test order returns to the default order which can be defined by the “**params.include**” annotation.

Reference: *For more in depth information about Multi Users go to Error! Reference source not found.on page Error! Bookmark not defined..*

6.2.3.2 Order Parameters Tabs

In order to change and correct the order of the parameters tabs, click on the “**Sort Sections**” button. A pop-up menu opens with two types of options:

1. **Predefined** – as defined by test author.
2. **Alphabetical** – alpha numeric sort by tab name.

6.2.3.3 Global Replace

In order to change the value of a parameter in several tests that belong to the same scenario, select the parent scenario of the tests, in the parameters area, the parameters appear for all the tests under the scenario.

Set the value of the parameter that requires changing, and press on the **"Apply for Scenario"** button.



Figure 4: Changing Test Parameters

The snapshot shows the tests parameters area when a scenario is selected. Note that the **"Apply for Scenario"** becomes enabled only when value of one or more parameters is changes.

There is a possibility that the operation will affect several tests, because of this a dialog opens and asks for user approval for the operation in order to commence.

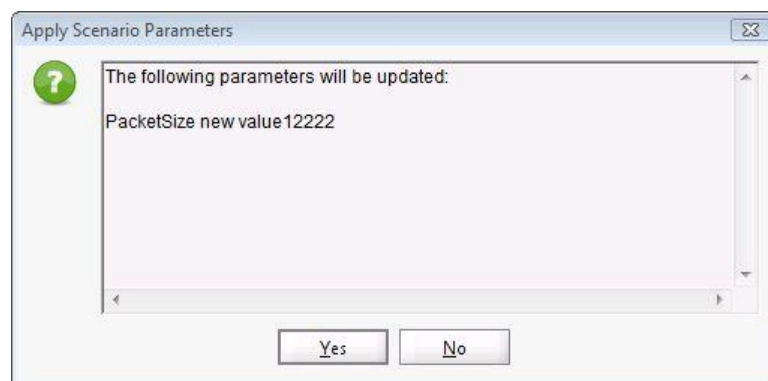


Figure 5: Applying Scenario Settings

6.2.4 Parameter Reference

The test parameters panel has been extended to allow setting of parameters values during runtime by reference to other JSystem components.

Parameters values can refer to:

- RunProperties properties.
- Summary properties.
- Sut file.
- Random value by range (for numerical values).
- Random value from group.

The parameter types that can be referenced are: Strings, File, Date, Integer, Long, Float, Double. Value parameter types such as: enum, or boolean cannot have reference in them.

The general syntax for reference is: **`${<Type>:<reference>}`**

6.2.4.1 Working with Parameter Values - Reference Example

In order to work with parameters values references, perform the following:

1. Write a test with parameter.
2. In JRunner select the test and add it to the scenario.
3. Open the parameters panel of the test.
4. Add a reference.

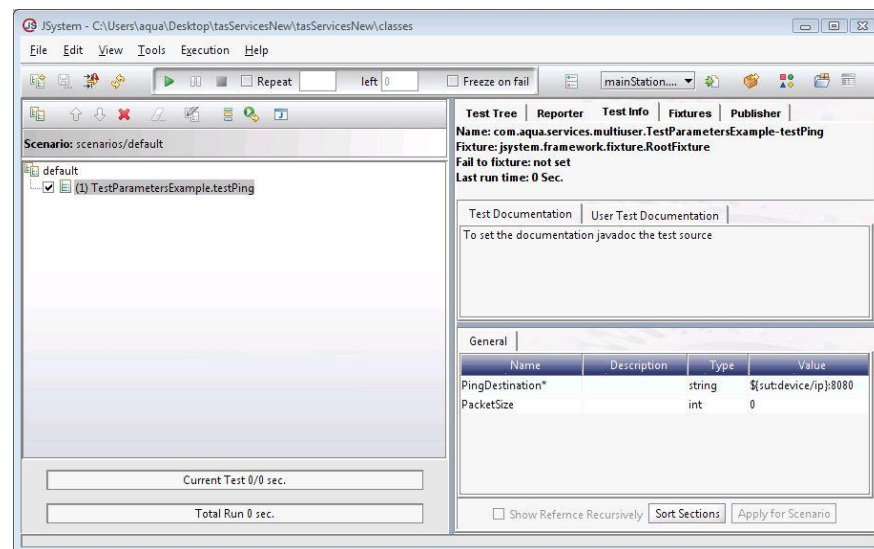
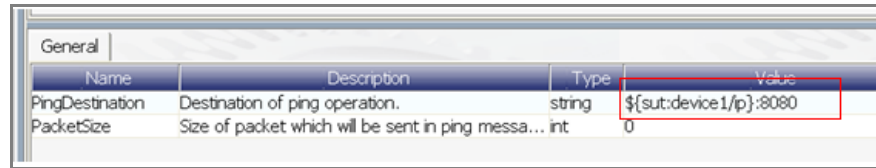


Figure 6: Adding a Reference

5. Zoom in:



Name	Description	Type	Value
PingDestination	Destination of ping operation.	string	<code>\${sut:device1/p}:8080</code>
PacketSize	Size of packet which will be sent in ping messa...	int	0

Figure 7: JSystem General Values

6. In this example we added a reference to the SUT file

At run time, before test is executed, the system will go to the SUT file, to the Xpath "**device1/ip**" and will replace "**`${sut:device1/ip}`**" with the value found in the SUT file.

6.2.4.2 RunProperties and Summary:

"**RunProperties**" and the "**Summary**" file can be accessed in the code as follows:

RunProperties.getInstance().setRunProperty(key,value).
Summary.getInstance.setProperty(key,value).

The values can be gathered later on by using the parameters:
`${run:<key>}` or **`${summary:<key>}`** accordingly.

6.2.4.2.1 Code Example

```
1 RunProperties.getInstance().setRunProperty("RunKey1","12").
2 Summary.getInstance.setProperty("SummaryKey1","C:\Program Files
  \").
3 JRunner parameter value:
4 ${run:RunKey1} and ${summary: SummaryKey1}
```

Table 3: Run Properties Code Example

6.2.4.3 SUT File:

The current SUT file tags can be accessed using the correct Xpath.
The syntax is **`${sut:<xpath>}`**

6.2.4.3.1 SUT Code Example

```
1 <sut>
2     <device1>
3         <class>sysobj.Device1</class>
4         <password>guy</password>
5     </device1>
6     <obj>enter user name</obj>
7 </sut>
```

Table 4: SUT Code Example

6.2.4.3.2 JRunner Parameter Values

`${sut:device1/password}`

`${sut:obj}`

6.2.4.4 Randomization

By Range - (for numerical parameters only) A number can be selected randomly within a given range. Syntax: `${random:<Low Value>:<High Value>}`|

6.2.4.4.1 Code Example

```
${random:1:15}
${random:125.66:130.12}
```

Table 5: Randomization Code Example

From Group - (for all parameters) a value will be selected randomly from a given group. Syntax: `${random:(<First Value>;Second Value>;Third Value...>)}`

6.2.4.4.2 Code Example

```
${random:(1;2;12;99)}
${random:("C:/Jsystem";"C:/Automation)}
```

Table 6: Randomization Code Example 2

6.2.5 Scenario Parameters

The user of the JRunner can define scenario parameters similar to test parameters definitions. This feature provides JRunner users with the ability to build reusable scenarios, and equip the JRunner with advanced features a tool for high end users that are not programmers.

6.2.5.1 Using Scenario Parameters

The following example demonstrates a scenario that validates the device version. The name of the scenario is **"validateVersion"**. The scenario exposes two parameters:

1. **machineIP** – IP address of the machine which version should be validated.
2. **fileToCheck** – The file on the remote machine which holds machine version.

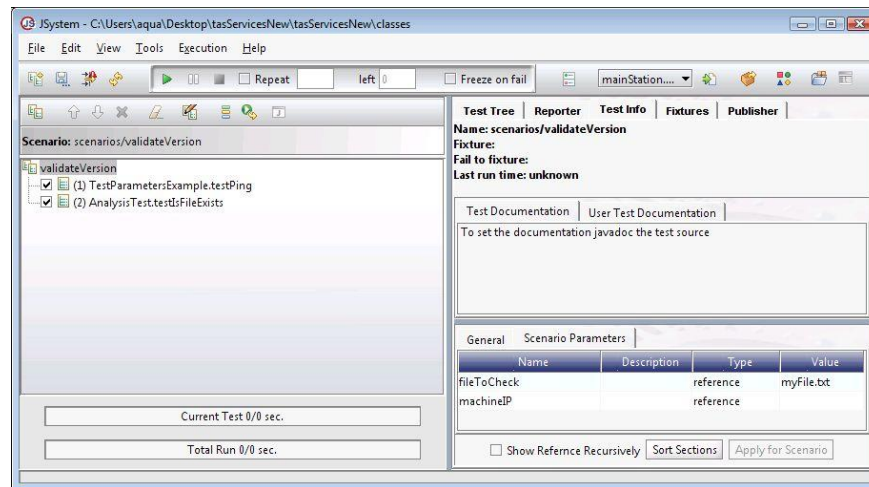


Figure 8: validateVersion scenario

The **"validateVersion"** scenario comprises of two tests, when the user selects the scenario and goes to the **"Test Info"** tab of the scenario there is a new tab, the **"Scenario Parameters"** this tab shows the parameters that the scenario exposes.

1. Create another scenario – **"validateSetup"** that uses the **"validateVersion"** scenario in order to validate the version of two machines in the SUT.

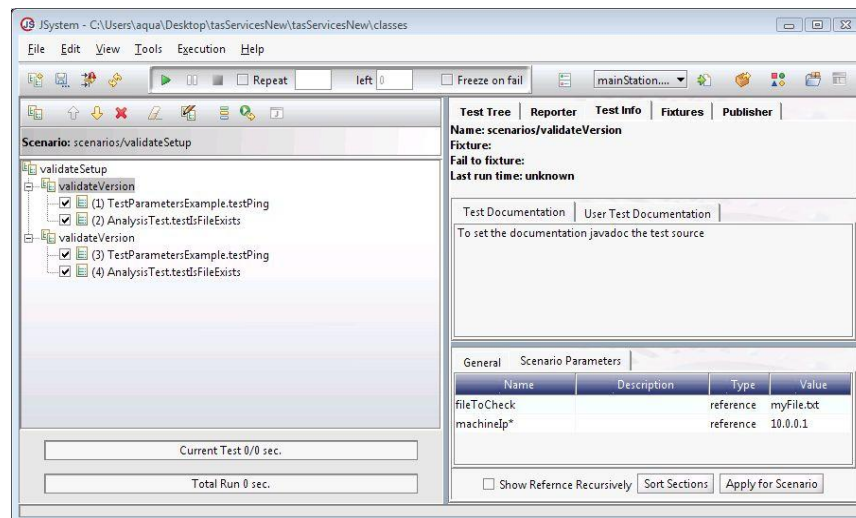
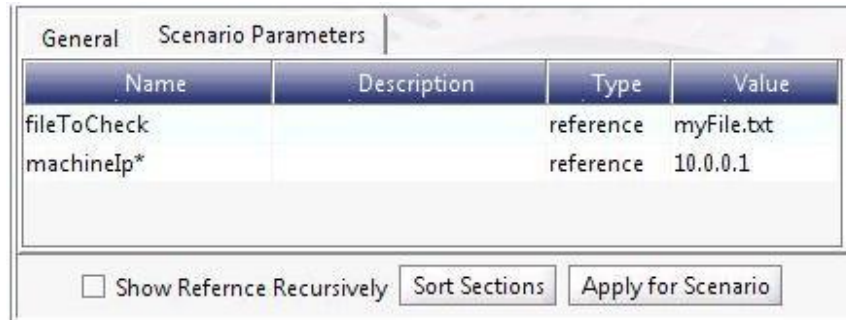


Figure 9: validateSetup scenario,

The displayed scenario uses the “**validateVersion**” scenario. By selecting the “**Test Info**” tab of an instance of the “**validateVersion**” scenario, the user can see scenario parameters and set their values.

2. In the example, set the “**machineIP**” of one instance to 10.0.0.1 and the second “**machineIP**” instance to 10.0.0.2.
3. After setting the scenario parameter value apply the operation by pressing on the “**Apply for Scenario**” button.



Name	Description	Type	Value
fileToCheck		reference	myFile.txt
machineIp*		reference	10.0.0.1

☐ Show Reference Recursively

Figure 10: Applying for Scenario

*Note: The user of the “**validateVersion**” scenario does not have to understand the tests in the scenario nor the parameters that the tests extract, all that must be understood is the parameters of the scenario.*

6.2.5.2 Defining Scenario Parameters

Once the scenario parameters have been set they must be defined by performing the following steps:

1. Select a scenario that requires a parameter definition.
2. Assume that the name of the parameter being added is the “**machineIP**”.
3. Now select the test in order to apply the scenario parameter on it, and add the scenario parameter in the test parameter value in the format **`${scenario:ParamName}`**.

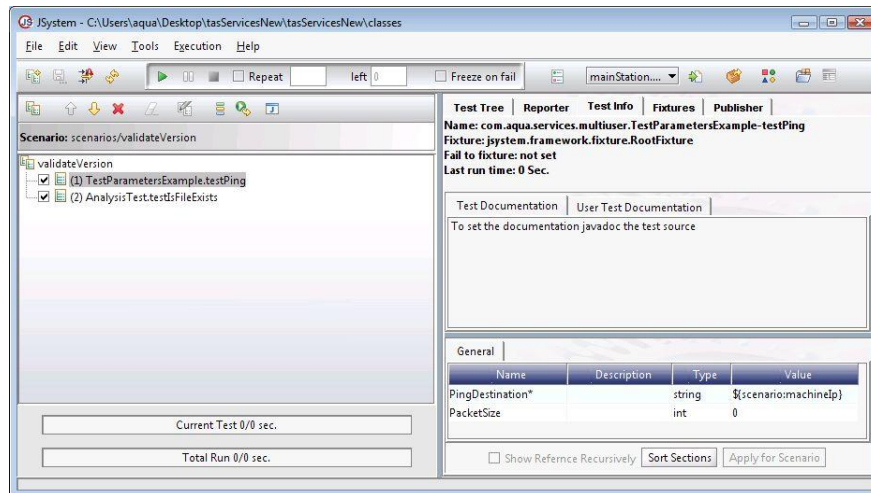


Figure 11: Defining Parameters

3. Zoom in.



Figure 12: Setting Values

4. Incorporating the scenario parameters in the scenario also defines the parameter.

6.2.5.3 Scenarios Parameters with Nested Scenarios

The behavioral characteristics of nesting several scenarios are as follows:

1. Take the “**validateSetup**” scenario and add it to a scenario called “**setupMaster**”

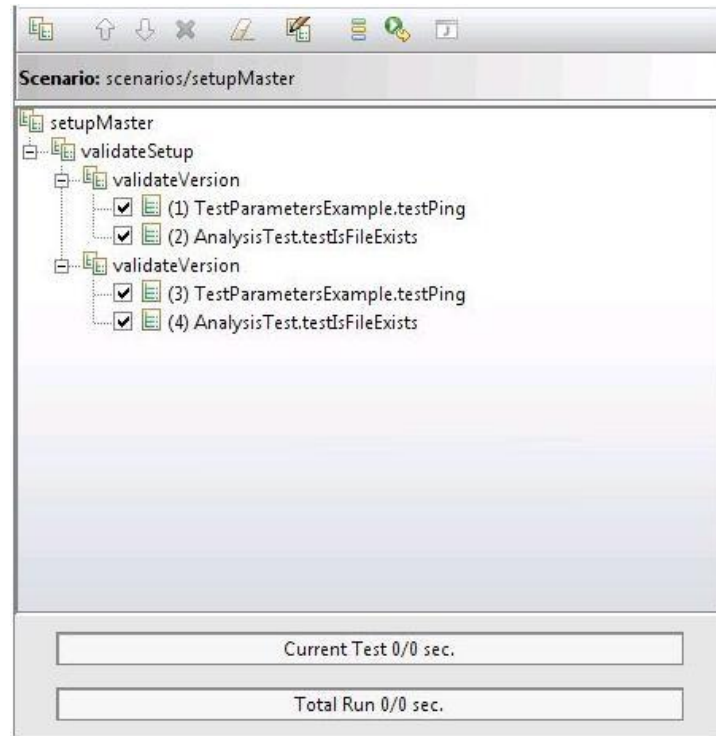


Figure 13: Nested Scenario Parameters

2. When selecting one of the instances of the “**validateVersion**” scenario the user can see that the values that were given to the parameters in the “**validateSetup**” scenarios are kept.

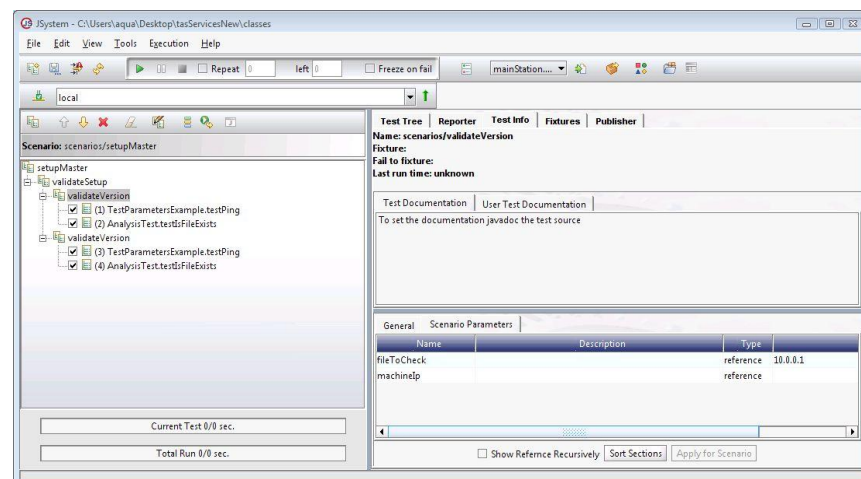


Figure 14: Validate Setup

3. Zoom in:

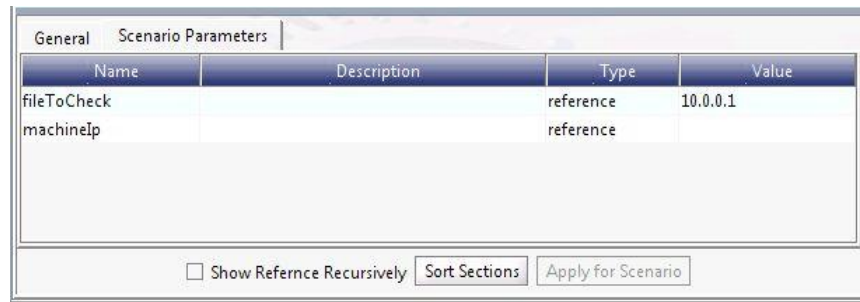


Figure 15: Scenario Parameters

4. The user can alter these values, by selecting the "**validateSetup**" as the root scenario; the original values of the parameters will not be changed.

6.2.5.4 Master Scenario

When nesting a sub scenario into a master scenario, the framework takes the default values of the scenario parameters from the higher level scenario under it, in which the user gave the parameter a value.

6.2.5.5 Default Parameter Values

In order to give a parameter a default value, select the scenario in which the parameter is defined, in our example, "**validateVersion**", now select the scenario "**Parameters**" tab and provide the parameter a default value.

Now, select one of the parent scenarios, for example, "**setupMaster**" right click on the scenario and select "**restore defaults**" menu item. If the user selects one of the instances of the "**validateVersion**" scenarios the values of the parameters are resorted to their default values.

6.2.5.6 Global Replace

A checkbox has been added to the parameters panel called "**Show Reference Recursively**". If it is not checked, only root tests scenario parameters will be shown. When checked, all Recursive sub tests parameters will be displayed. When working with recursive checked, and applying a change to Scenario Parameters, all sub scenarios will be changed. This does not apply to regular parameters.



Figure 16: Show Reference

6.2.6 Flow Control Enabled Scenario

The **"Flow Control"** feature expands the capabilities of a test scenario to include a GUI based scripting mechanism and complex flow control, providing the user with the ability to monitor the test flow within a scenario.

There are three kinds of flow control types the test operator can implement into a scenario, **"for"**, **"if"**, and **"switch"**.

6.2.6.1 Enabling Flow Control

In order to activate the flow control check the **"Flow Control Toolbar"** item in the **"View Menu"** **"Toolbars"** > **"Flow Control Toolbar"** as it appears in the illustration.

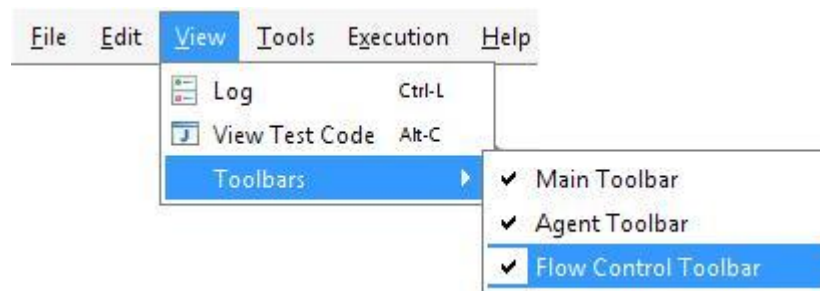


Figure 17: Flow Control Menu

6.2.6.2 Return Parameters

JSystem Automation Framework supports tests that include return parameters, these **"return Parameters"** are return to the scenario, and are used as the input for the rest of the test or scenario execution.

In order to add a return parameter the user should add the following elements to the code:

- **"import jsystem.framework.TestProperties;"**
- **@TestProperties** annotation for each of the tests, with a list of **"returnParam"** strings.
- Setter and getter methods for each of the return parameters, starting with **"set"** and **"get"/"is"** respectively.

The following example shows a test that includes a return parameter.

```
01 package com.aqua.sanity;
02
03 import jsystem.framework.TestProperties;
04 import com.aqua.general.JSysTestCase;
05
06 public class FlowControlFunctionality extends JSysTestCase {
07     private int NumberOfFiles = 42;
08     private int ExpectedNumberOfFiles = 3;
09     @TestProperties(returnParam={"ExpectedNumberOfFiles","NumberOfFiles"})
10     public void testNumberOfFiles() {
11         assertEquals(getNumberOfFiles(), getExpectedNumberOfFiles());
12     }
13
14     public int getNumberOfFiles() {
15         return NumberOfFiles;
16     }
17
18     public int getExpectedNumberOfFiles() {
19         return ExpectedNumberOfFiles;
20     }
21
22     public void setExpectedNumberOfFiles(int expectedNumberOfFiles) {
23         ExpectedNumberOfFiles = expectedNumberOfFiles;
24     }
25
26     public void setNumberOfFiles(int numberOfFiles) {
27         NumberOfFiles = numberOfFiles;
28     }
29 }
```

Table 7: Flow Control Code Example

6.2.6.3 “For” Flow Control

The **“for”** flow control - creates a test loop above the test layer, according to a list of user defined values. When pressing on the **“for”** button, a **“For”** branch is added to the scenario tree, into it the user can add tests, scenarios, and other flow control objects.

The example shows a basic scenario with a **“for”** loop, and a test inside it.

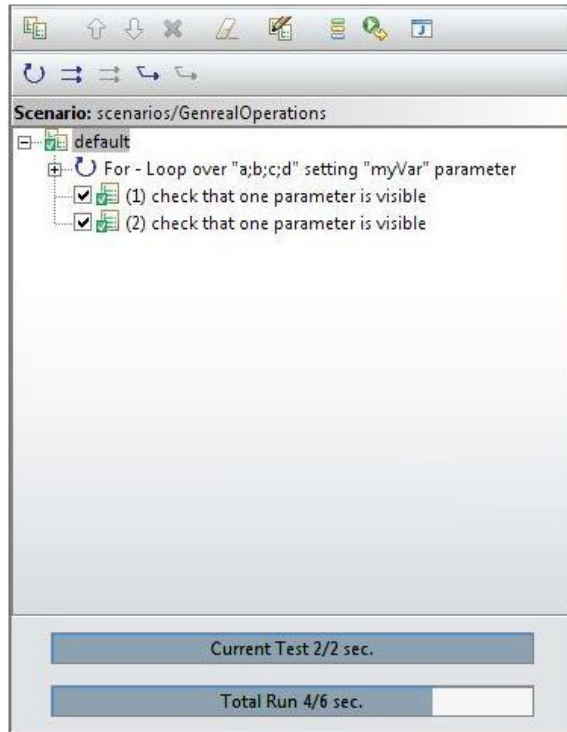


Figure 18: Creating Flow Controls

1. The Flow Control Toolbar positioned at the top of the scenario panel.
2. The **“For”** branch inside the scenario, contained within the tests.
3. In the right side - the **“list”** and the **“loop value”** parameters that appear in the **“Test Info”** tab, **“General”** sub-tab.
- 4.

6.2.6.3.1 For Object Execution

When a scenario with a **“for”** object is executed, all of the tests in the loop are executed **“N”** times, according to the number of values in the **“list”** parameter, a semi-column-separated list of values.

The current value from the list is set into the **“loop value”** at every iteration, which can be used as a parameter inside the loop. In the example above, the test is executed 4 times, each time setting the value into `${myVar}`.

Note: In order to use the loop variable in tests inside the loop, set one of the test's parameter to be the loop variable. In the example, the value of a parameter in the loop is set to `${myVar}`.

6.2.6.4 “If” Flow Control

The “if” flow control - creates a conditional statement that changes the execution flow of the scenario. By pressing on the “if” button, a branch is added to the scenario tree, with an additional “else” branch inside it. An additional “else if” branches can be added into an “if” branch, in order to create a more complex execution flow.

1. Each “if” branch must have an “else” branch inside it, it can be empty, but cannot be deleted.
2. The “if” condition becomes useful when combined with a new “**return parameters**” mechanism (as described above).

For example, a scenario containing an “if” branch, one “else if”, and the “else” branch appears as follows.

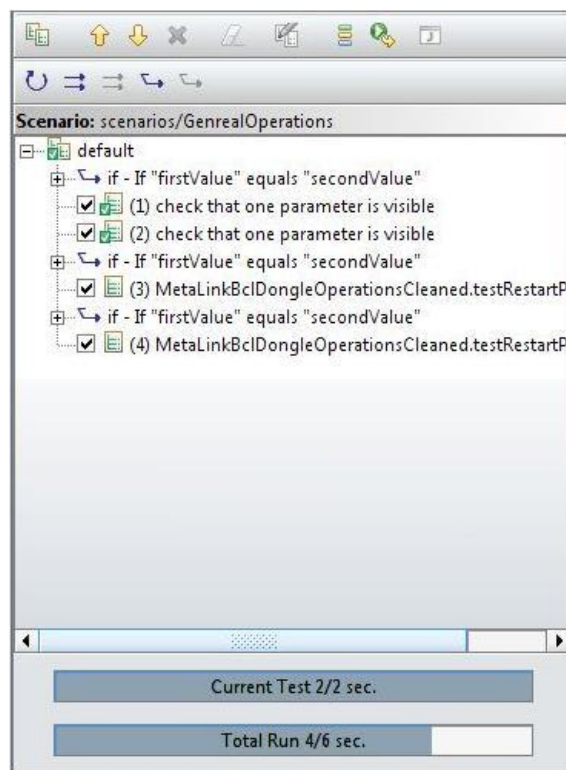


Figure 19: Adding an “If” Flow Control

Important Considerations - The place of the comment is used to show the condition of each of the “if” objects.

3. Each test has a list of “**Return parameters**” at the top of the “**Test Info**” tab.
4. The ability of a test to refer to a “**return parameter**” is performed by using **`${<parameter name>}`**.

Note: Known limitation – The parameter name must use a capital Letter at the beginning of the word (Even when the actual parameter name uses a lower case character).

6.2.6.5 “Switch” Flow Control

The **"switch"** flow control - similar to an **"if"**, the **"switch"** object can change the execution flow. By pressing on the **"switch"** button, a branch is added to the scenario tree, with an additional **"default"** branch inside it.

Additional **"case"** branches can be added to a **"switch"** branch. Each **"switch"** branch must have a **"default"** branch inside it, which can be empty, but cannot be deleted.

The example illustrates the adding of a **"switch"** control to a scenario.

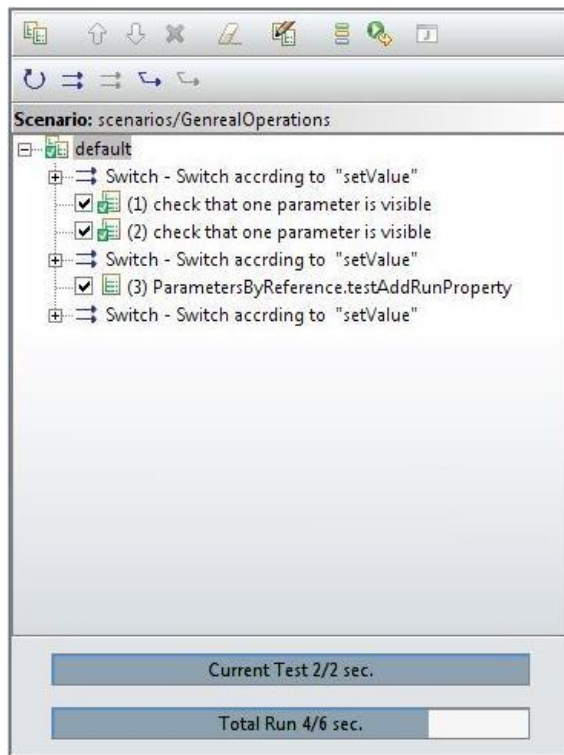


Figure 20: Adding a Switch Flow Control

6.3 Working with Fixtures

Fixtures can be assigned to a test programmatically or by assigning a fixture to a scenario. In order to assign a fixture to a scenario, select the fixture in the **"Test Tree"** and press on the ➡ **"Add test to scenario"** button. When assigning a fixture to a scenario, the fixture is automatically assigned to all the tests in the scenario.

***Note:** Assigning a fixture to a test by assigning it to a scenario overrides programmatic assignment. Only one fixture can be assigned to a scenario. Unlike tests, fixtures do not support parameterization.*

6.3.1 Working with Sub-Scenarios

When using the scenario studio the user can build a compound scenario. To add a sub-scenario to the root scenario or to one of the sub scenarios of the root scenario perform the following steps:

1. Select the test in the scenario after which the sub scenario will be added.
2. Select the scenario you want to add in the **"Test Tree"**.
3. Press on the **"Add test to scenario"** button.

6.3.1.1 Working with Multiple Instances of a Sub Scenario

A scenario can be added several times as a sub-scenario. When editing a sub scenario note the following:

1. Changing the test order or removing or add tests to scenario the changes made to the sub scenario will affect all sub scenario instances.
2. Editing test parameters, changes will affect only the sub scenario instance that is working on.
3. Changing user documentation, meaningful name and or comment of a sub scenario only affects the instance that is being working on.

6.3.2 SUT Editor

Before running a scenario in a new environment, the SUT file for the specific environment has to be selected, or adjustments to an already existing SUT file must be made.

***Reference:** For more information about SUT configuration refer to Error! Reference source not found. on page Error! Bookmark not defined..*

6.3.2.1 Changing SUT File

The list menu of all the SUT files that were written by the authors of the automation project can be seen in the SUT drop down list. In order to change SUT files, select the new SUT file from the drop down list.



Figure 21: Changing SUT File Settings

6.3.2.2 Editing SUT File

To edit the SUT file, press on the **"Edit Sut"** button.



Figure 22: Editing SUT Files

When pressing on the button the default editor opens.

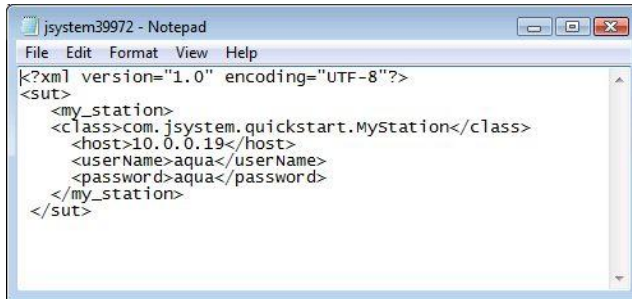


Figure 23: Default Editor

6.3.3 Changing the Default SUT Editor

In order to change the default SUT editor set the property **"xml.editor"** in the **"jsystem.properties"** file. The property should be set with a path to the editor that is identifiable by the operating system.

For example `xml.editor= C:\\Program Files\\Vim\\vim71\\gvim.exe`

6.3.3.1 Developing a New SUT Editor

In order to implement a new SUT editor, write a java class that implements the **"jsystem.framework.sut.SutEditor"** interface and sets the **"sut.editor"** property with full class name.

For example: **"sut.editor= jsystem.treeui.suteditor.TabbedSutXmlEditor"**

In order to view an example of a custom SUT editor, go to the tabbed SUT editor:

"jsystem.treeui.suteditorTabbedSutXmlEditor"

6.4 Tabbed SUT Editor

The tabbed SUT editor is a customizable editor, in which the programmer can control the properties that are exposed to the JRunner user.



Figure 24: Tabbed SUT Editor

When working with the tabbed SUT editor, each system object has its own tab, the user can edit only the SUT entries that the SUT author has decided to expose. In addition to each entry the author can add a description that explains the purpose of each entry.

6.4.1 Configuring JRunner to Work with Tabbed SUT Editor

JRunner automatically identifies when a SUT file can be opened with the tabbed SUT editor. In order to signal the JRunner to open the SUT file with the tabbed SUT editor add the property **edit= "enable"** to the entries that require exposing to the user. The JRunner identifies edit="enable" and automatically opens the SUT file with the tabbed SUT editor.

The tabbed SUT editor automatically configures the correct dialog dimensions; in order to override the default settings, go to the control dialog settings to affect size use the width and height properties.

6.4.1.1 Code Example

```
1 <sut width="320" height="350">
2   <station>
3     <class>com.aqua.sysobj.conn.WindowsDefaultCliConnection</class>
4     <user1 edit="enable" description="telnet user name">
5       simpleuser</user1>
6   </station>
7 </sut>
```

Table 8: Configuring JRunner SUT Code Example

6.4.2 System Object Browser

The purpose of the System Object browser feature is to allow the JRunner user (not a Java programmer) to implement scenarios without having to write JSystem tests.

Using Java introspection the System Object browser generates tests that expose System Object functionality, once the tests are generated, the user can select these tests and compose a scenario from these tests.

When working with the System Object browser it is assumed that system objects that define the model of the SUT are written correctly and work well.

6.4.2.1 Activating the System Object Browser

1. Press on the **"System Object Browser"** button.



Figure 25: System Object Browser

2. The system object browser dialog will open with a list of all the system objects which are defined in the current SUT.



Figure 26: System Object Browser Selection

3. Select the system object for the JRunner test generation from the drop down menu.

Once the system object is selected, the JRunner starts the introspection process.

During the process a progress dialog appears.

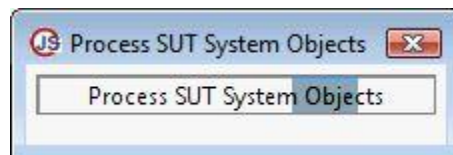


Figure 27: SUT Progress Bar

4. Once the process is complete, a new “**autogen**” package appears, under it, are sub packages with system object name.

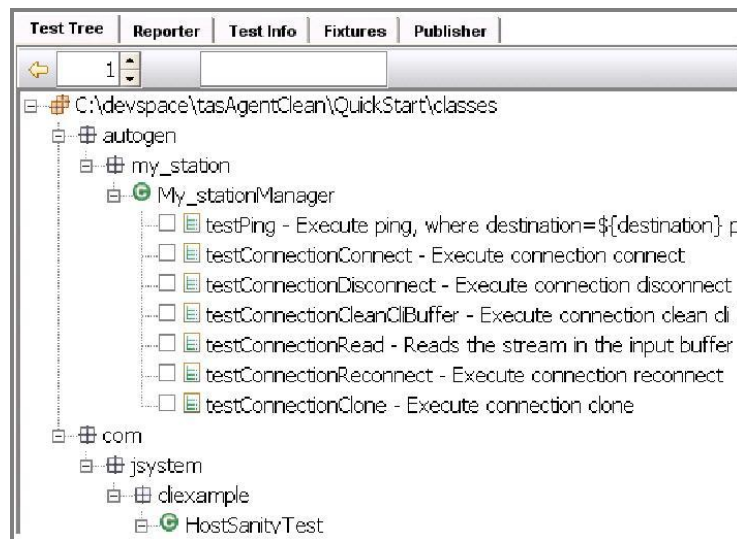


Figure 28: New Package Autogen

The system object browser generates a test method for each public method the system object exposes.

5. Now select the tests and build the scenario.

6.4.2.2

6.4.2.3 System Object Browser Generated Code

In many cases once the code has been generated, manual changes need to be made to the code, in addition the system object might change and the auto-generation might be re-activated. In order to avoid conflicts between auto generated code and manually written code, the System Object Browser module generated two classes for each system object:

1. Abstract base class that includes all the tests and class members.

Note: The documentation from the system object is imported to the test class. This class should not be altered by the user manually.

2. Concrete class that extends the base class. By default the concrete class is empty. Manual changes that need to be made should be directly made on this class.
3. When the system object is changed by the developer and the classes are regenerated, only the base class is overridden, by the JRunner, the extending class is not altered.

6.5 JRunner

The JRunner is the second module in JSystem; this is the module that is responsible for executing the scenario, helping the user to follow the execution, and at the end of the execution, helping the user to analyze the results.

6.5.1 Scenario Execution

The following section contains a list of scenario execution operations with a short technical description of each operation.

6.5.1.1 Run

In order to run a scenario, press on the run button, or activate the play menu item (Execution → Run).



Figure 29: Scenario Run Button

When run is activated, a new JVM for scenario is spawned from JRunner JVM. The new JVM activates an ANT interpreter which executes the scenario script.

6.5.1.2 Stop

In order to stop scenario execution, press on the stop button, or activate the stop menu item (Execution → Stop).

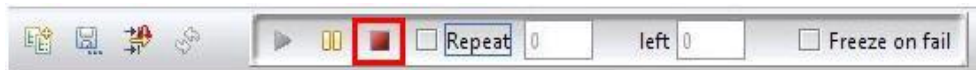


Figure 30: Scenario Stop Button

6.5.1.2.1 The Stop Operation

When the stop operation is activated the remote JVM that executes the scenario is signaled to stop and terminate. Termination is immediate, the tear down method of the currently active test is not executed and system objects are not signaled to release resources.

6.5.1.3 Pause

In order to pause the scenario execution, press on the pause button, or activate the pause menu item (Execution → Pause).



Figure 31: Scenario Pause Button

When the pause operation is activated a pause signal is sent to the remote scenario execution JVM, and scenario execution is paused.

In order to implement the pause functionality, the framework uses the reporter service. When the pause signal arrives to the remote tests JVM, a flag is raised, the next time a report message is logged, the reporter identifies that the pause flag is up, and goes over all the system object in the framework repository and signals them to pause by activating their "**pause()**" and enter the execution thread to the "**wait**" state.

When the run button is pressed again, the remote test JVM receives the run signal, the "**resume()**" method of all system objects in the system is invoked and the execution thread is resumed.

6.5.1.4 Freeze on Fail

In order to signal the JRunner to freeze the execution on test failure, check the "**Freeze on fail**" check box:



Figure 32: Freeze on Fail

When a test fails a dialog opens and the JRunner waits for user input to continue execution.

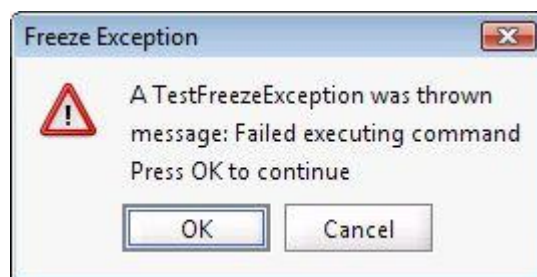


Figure 33: Freeze Exception Dialog

6.5.2 Fixtures

The JRunner module in JSystem, allows the user to track fixture navigation during scenario execution, and to manually trigger the navigation to fixtures.

In order to track fixture navigation during scenario execution, select the fixtures tab.

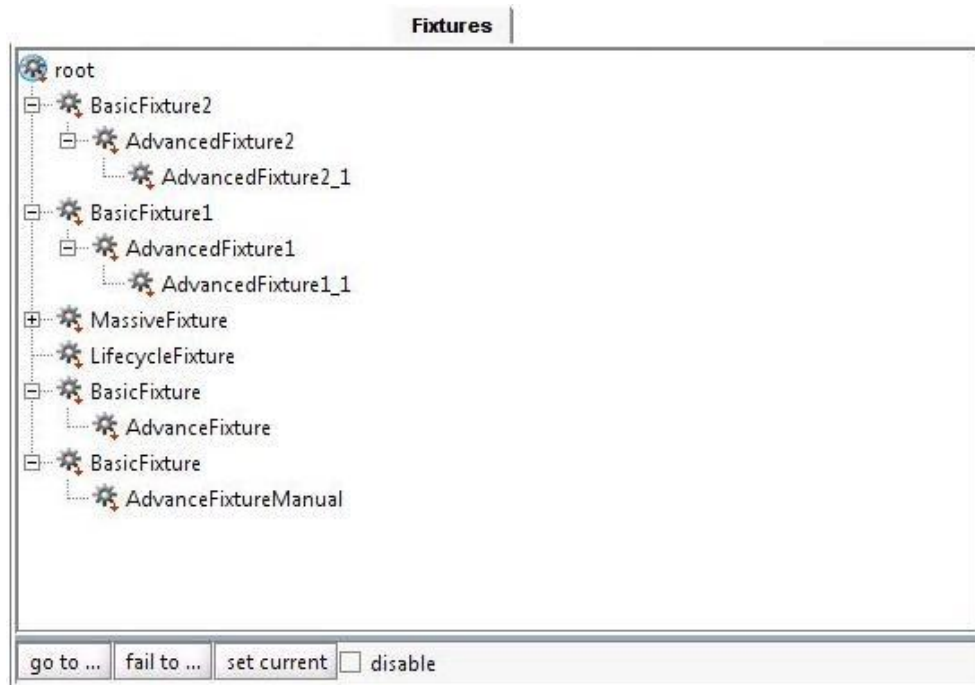


Figure 34: Fixture Services

The Fixtures tab in the JRunner displays the fixtures that are defined in the system and their resulting hierarchical structure.

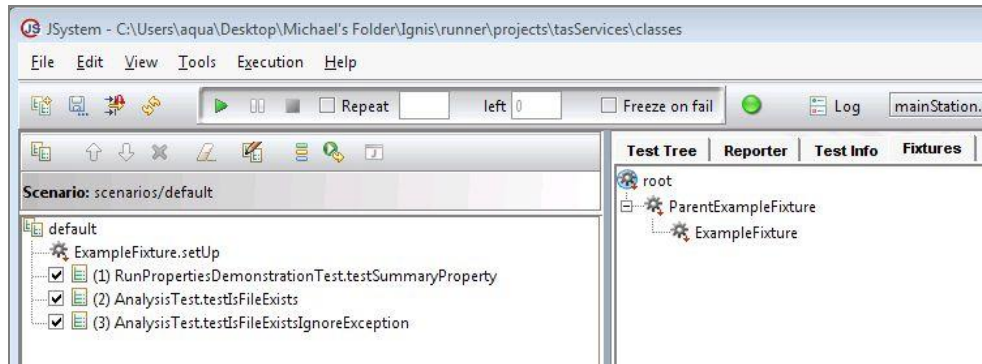




Figure 35: JRunner Fixture View


During the execution of a scenario, the JRunner graphically navigates down the fixtures tree inside the fixtures tab. The current fixture is highlighted in blue.

6.5.2.1 Manual Navigation to a Fixture

In order to manually navigate to a fixture press the  "Go to" button.

To navigate through "failTearDown" methods, press the  "Fail to" method as appears in the JRunner image.

6.5.2.1 Setting the Current Fixture without Navigation

In order to manually set the current fixture without having to navigate to it, select the fixture and press the  "set current" button.

6.5.3 Analyzing an Execution with the JRunner Log

During scenario execution, report events are sent to the JRunner by the reporter service. The user can follow the progress of scenario execution by tracking the content in the “**Reporter**” tab, or see a full execution log by opening the JRunner log.

Reference: For more information regarding the JReporter service refer to Error! Reference source not found. on page Error! Bookmark not defined..

6.5.3.1 Reporter Tab

In order to view the execution progress, select the “**Reporter**” tab.

Test Tree	Reporter	Test Info	Fixtures	Publisher
Commands				
				Status
Start: My_stationManager.testPing				
(1)Steps in test: Execute ping, where destination=\${Destination} packetSize=0 :				✓
Params:				
PacketSize=0				✓
18:07:03: setUp execution				
18:07:03: In init method				✓
18:07:03: Init cli, host: 10.0.0.19				✓
18:07:13: The test was interrupted by the user				✓
Start time: Mon May 05 18:07:03 IDT 2008				✓
End time : Mon May 05 18:07:13 IDT 2008				✓
Test running time: 10 sec.				✓
End: My_stationManager.testPing				✓
Start: My_stationManager.testPing				
(2)Steps in test: Execute ping, where destination=\${Destination} packetSize=0 :				✓
Params:				
PacketSize=0				✓
18:07:17: Fixture: InstallPatch setUp				
18:07:17: In init method				✓
18:07:17: Init cli, host: 10.0.0.19				✓

Figure 36: Reporter Tab Progress

The JReporter table shows the following information:

1. Test documentation
2. Test parameters
3. Test start & end time
4. Title of log message + message status.

6.5.3.2 JRunner Log

In order to see the full execution log press on the “**Log**” button or go to View→Log



Figure 37: JRunner Log

The default browser is opened in the HTML JReporter default.html page.

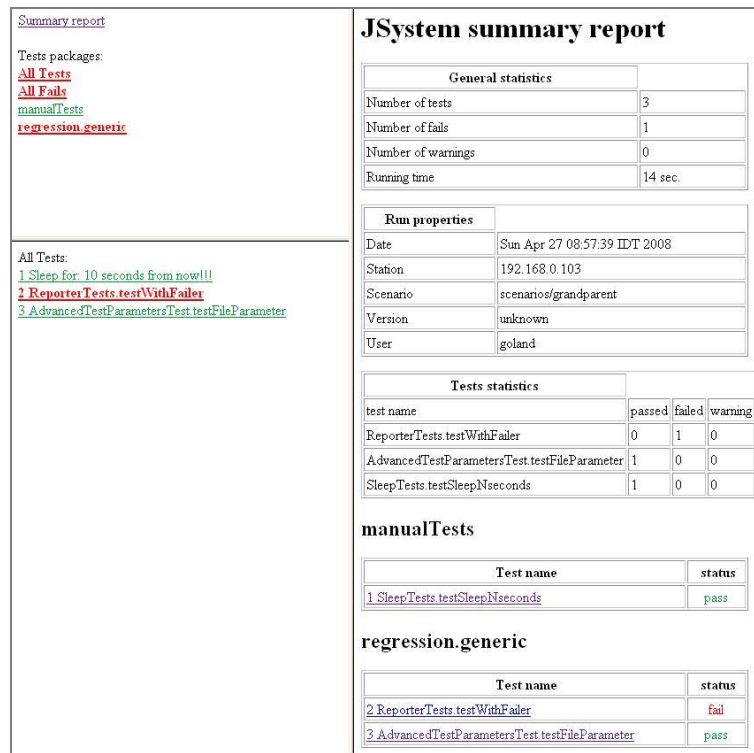
The image shows a screenshot of the JSystem summary report HTML page. The page is divided into two main sections. The left section contains a 'Summary report' with links for 'All Tests', 'All Fails', 'manualTests', and 'regression.generic'. Below this, it lists 'All Tests' with three items: '1 Sleep for 10 seconds from now!!!', '2 ReporterTests.testWithFailer', and '3 AdvancedTestParametersTest.testFileParameter'. The right section is titled 'JSystem summary report' and contains several tables. The first table is 'General statistics' with rows for 'Number of tests' (3), 'Number of fails' (1), 'Number of warnings' (0), and 'Running time' (14 sec). The second table is 'Run properties' with rows for 'Date' (Sun Apr 27 08:57:39 IDT 2008), 'Station' (192.168.0.103), 'Scenario' (scenarios/grandparent), 'Version' (unknown), and 'User' (goland). The third table is 'Tests statistics' with columns for 'test name', 'passed', 'failed', and 'warning'. It lists three tests: 'ReporterTests.testWithFailer' (0 passed, 1 failed, 0 warning), 'AdvancedTestParametersTest.testFileParameter' (1 passed, 0 failed, 0 warning), and 'SleepTests.testSleepNseconds' (1 passed, 0 failed, 0 warning). Below these tables are two more sections: 'manualTests' and 'regression.generic', each with a table showing test names and their status (pass or fail).

Figure 38: JSystem Summary Report

The output of the JSystem’s HTML JReporter is inspired by the JUnit report.

The index page is divided to three sections:

1. The top left region of the user interface shows a summary of the tests by java package.
2. Beneath it on the left hand side is a list of all tests, together with their status.
3. On the right side the user can see summary of scenario execution.

The log pages can be found under “**runner path→log→current**”. In order to configure the HTML reporter folder update the property “**htmlReportDir**” directory in the JSystem properties file, as appears in the example “**c:\\jssystemLog**”.

Note: The “*current*” property is automatically appended to the path.

<p>Summary report</p> <p>Tests packages:</p> <p>All Tests</p> <p>All Fails</p> <p>manualTests</p> <p>regression.genetic</p>	<p>(3)Steps in test: AdvancedTestParametersTest.testFileParameter :</p> <p>Params:</p> <p>CompareByModifiedDate=false</p> <p>CompareByCreationDate=false</p> <p>CompareBy=content</p> <p>IgnoreFileNameCase=false</p> <p>DemoType=hide</p> <p>Binary=false</p> <p>File=</p> <p>Date=</p> <p>File2=</p> <p>CompareBySize=false</p>
<p>All Tests:</p> <p>1 Sleep for: 10 seconds from now!!!</p> <p>2 ReporterTests.testWithFailer</p> <p>3 AdvancedTestParametersTest.testFileParameter</p>	<p>08:57:58: setUp execution</p> <p>08:57:58: tearDown execution</p> <p>Start time: Sun Apr 27 08:57:58 IDT 2008</p> <p>End time : Sun Apr 27 08:57:58 IDT 2008</p> <p>Test running time: 0 sec.</p>

6.5.3.3 Init Reports

6.5.3.4 Log Backup

6.5.3.5 Additional Reporter Configuration

- Note: The browser properties work only on Linux machines.**

6.5.4 JAR List

In order to see the list of “**library jars**” that the JRunner uses, their order and version use the JAR List dialog. Tools→Show library jars.

In order to open the “**JAR List**” dialog select from the tools menu: Tools→Show Jars List.

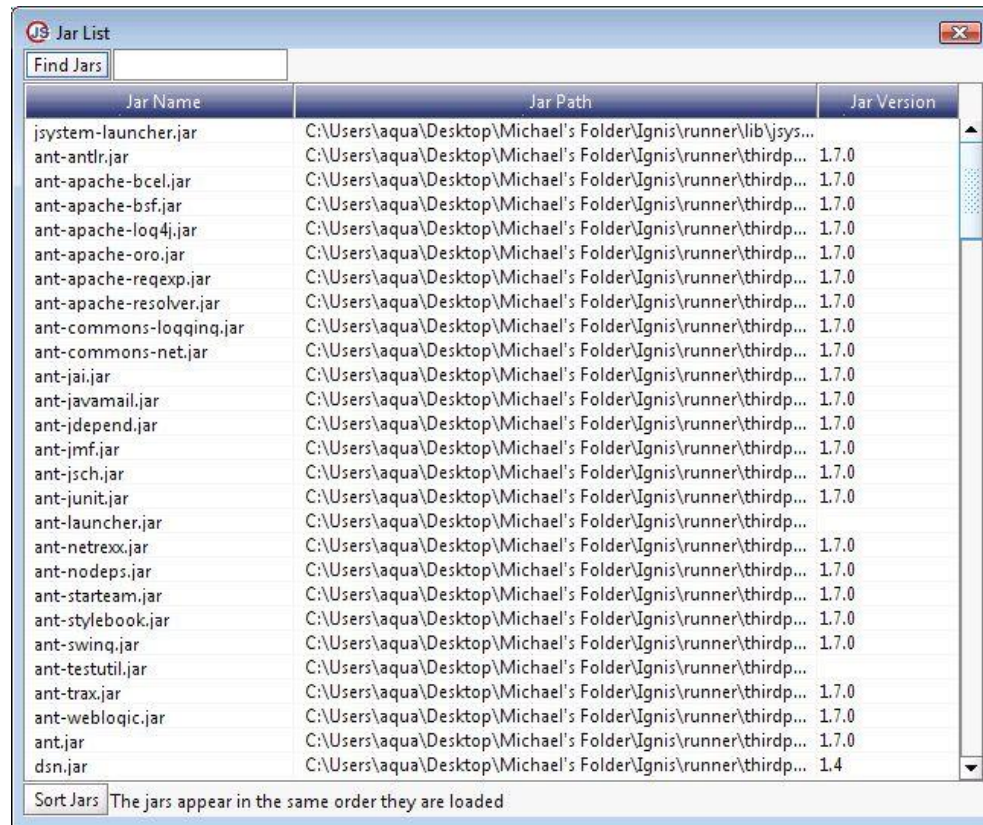


Figure 40: JAR List

The dialog shows a table of all the jars that JRunner uses in the class patch. By default the dialog shows the jars in their “**classpath**” order. The table displays the name of the jar, full path in file system and jar version.

6.5.4.1 Finding a Jar

In order to find a specific jar in the jars list, enter the **"jar name"** (or part of the name) in the find jar text input field and then press on the **"Find Jars"** button.

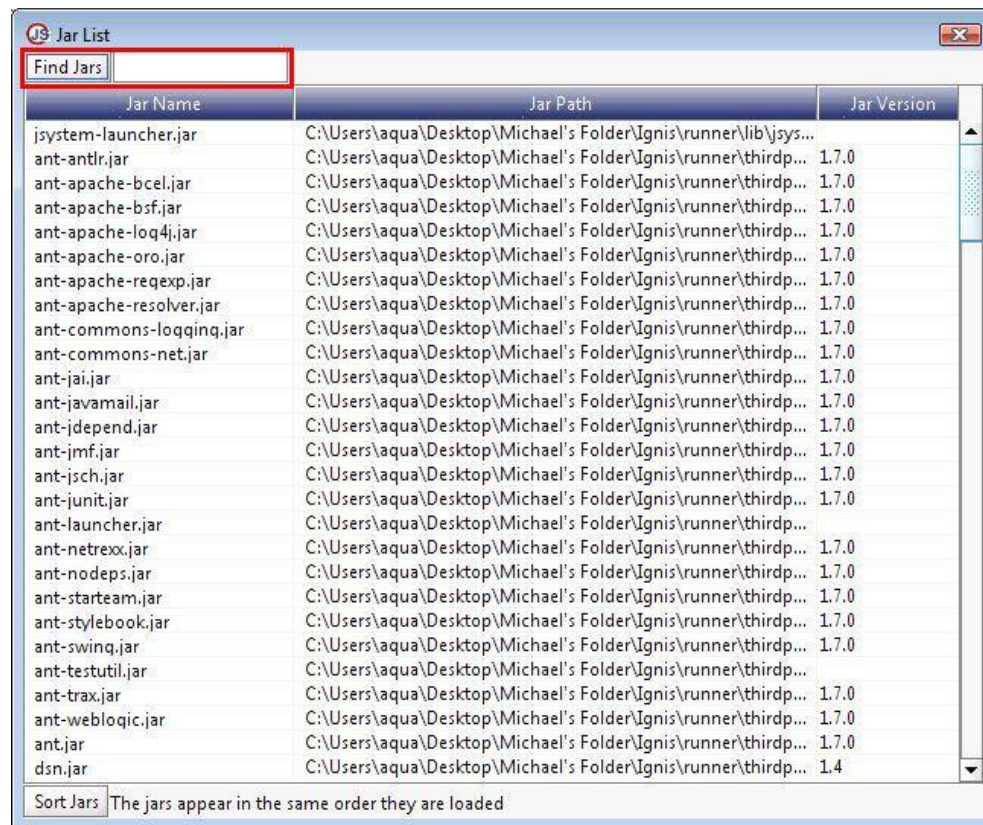


Figure 41: Finding JAR

Note: Matching jars are moved to the top of the list and marked in bold.

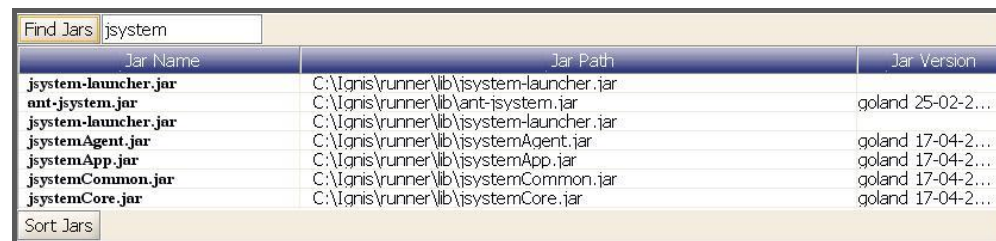


Figure 42: Matching JAR's

Another method can be used in order to find jars in the list, by sorting the list. In order to sort the list, press on the **"Sort Jars"** button. To the right of the button the jars order results can be viewed in the text field.

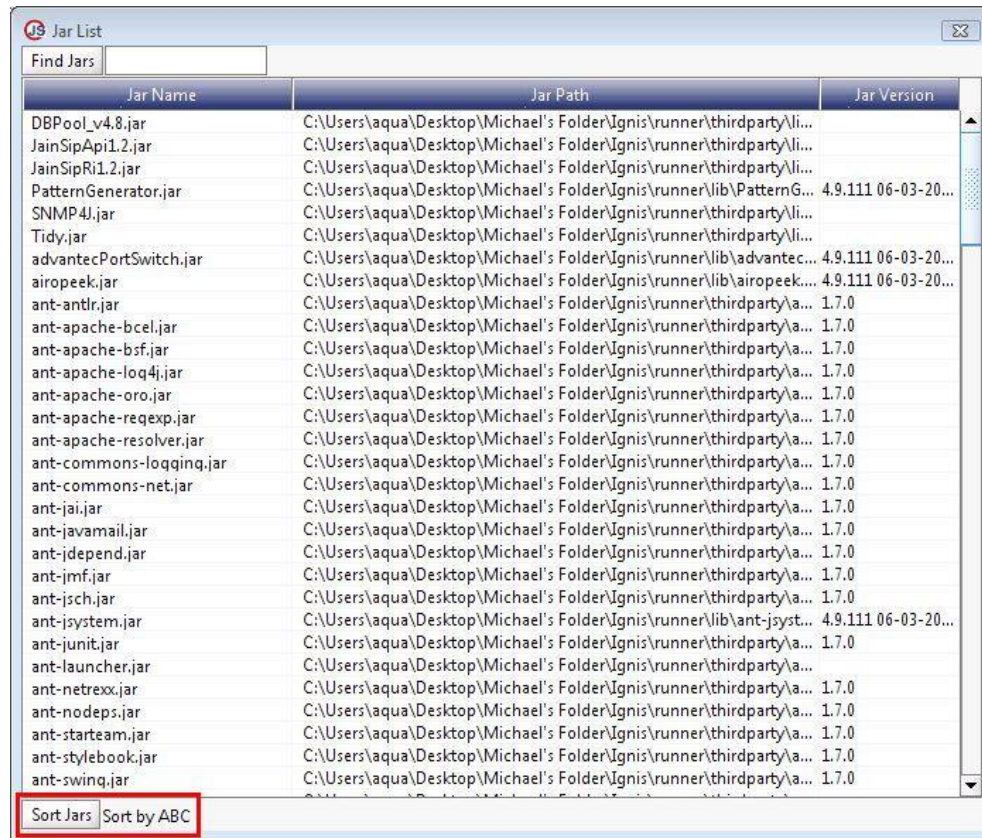


Figure 43: Sorting JAR Lists

6.5.5 JSystem Properties Dialog

The JSystem “**Properties Dialog**” enables the user to configure the JSystem JRunner preferences.

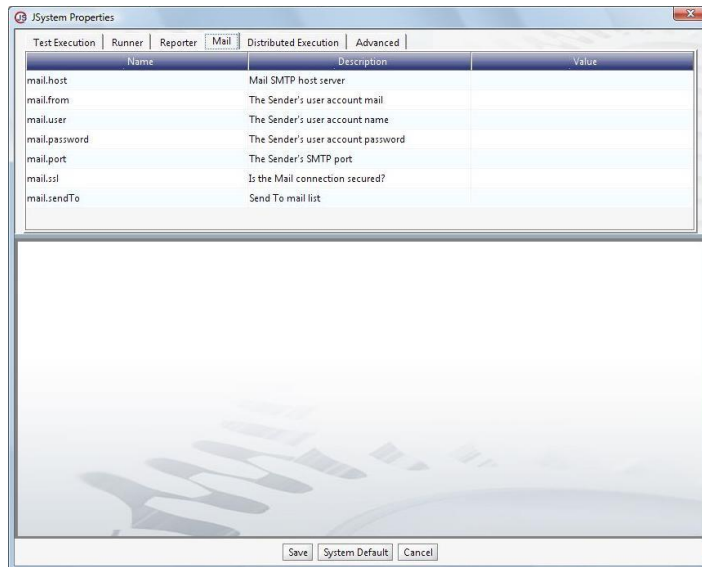


Figure 44: JSystem Properties Dialog

6.5.6 Using the JSystem Properties Dialog

In order to open the JSystem properties dialog from the JRunner, go to Tools --> JSystem Properties Dialog:

6.5.6.1 JSystem Properties Table

- **Property** - The property name (Taken from the Framework Option Enum)
- **Description** - A text field describe shortly the current property
- **Value** - An editable field allow the user to edit the property value.

Each property has its own editor make it easy to use for the user.

In general there are 3 different types of editors:

- **List box** - Enable the user to choose the desired value from a list of options.

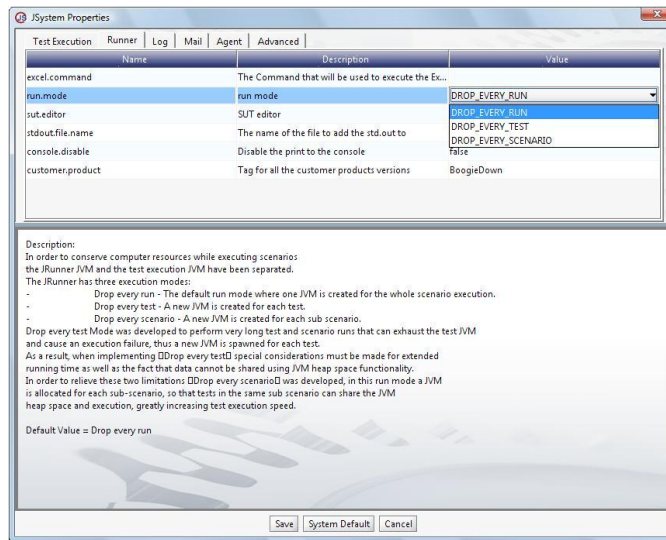


Figure 45: Properties Dialog Runner Tab

- **Text field** - To allow the user to type in the desired value

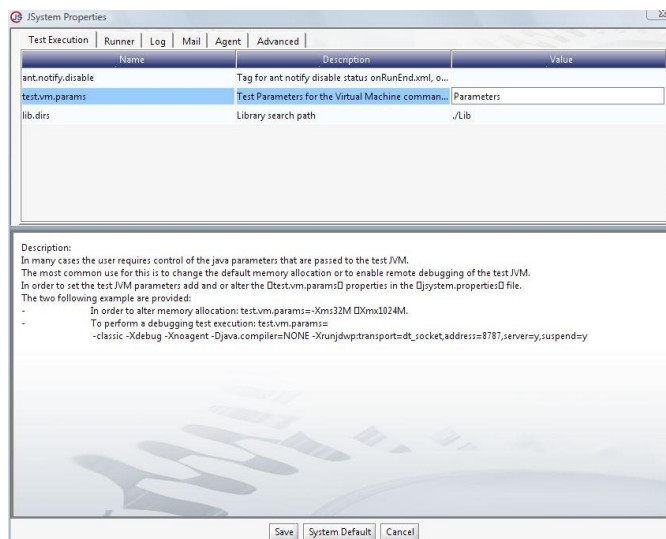


Figure 46: Properties Dialog, Setting the Parameters

- **File Chooser** – In order to allow the user to browse and select files or directories.

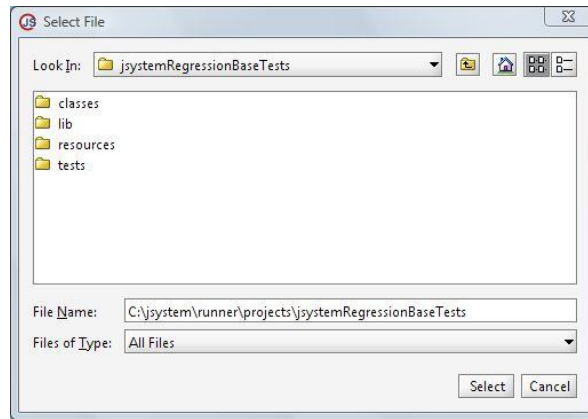


Figure 47: Properties Dialog - Selecting a File

- **Default Value** - This field present the default value for the current property.

The default value is used in the following cases:

- When the value field is empty.
- When the user press "**Restore System Default**".
- **Long Description** - A detailed description of the current property.

The long description provides an in-depth description of the selected property, usage, purpose, including examples if needed and a detailed explanation of the value options.

6.5.7 Operations

The JSystem properties dialog supports the following operations:

Save - Save all changes, and close the dialog.

The properties are saved into a properties file named "**jssystem.properties**" (In the JRunner installation folder).

Not all of the properties presented in the dialog are saved into the file, only properties that have been change. Some of the properties changes do not take effect until the JRunner is restarted.

When the user presses the "**save**" button, JSystem performs an automatic check in order to see if properties have been changed that require a JRunner restart. In the case of a restart, a pop up message is displayed to notify the user that some of the changes will not take effect until the JRunner is restarted. The user then can choose whether a restart of the JRunner is required immediately, or if can wait until a later time.

Restore Defaults - This operation will restore all of the default properties.

The "**default properties**" refers to all of the properties saved into "**jssystem.properties**" file under the installation JRunner directory.

In the event that the user chooses to restore the defaults, the "**jssystem.properties**" file will be deleted from the system file and will be recreated during the JRunner reload, with the system default values.

Note: By reopening the "jssystem.properties" file, most of the properties will not appear in the file, because only the properties that are change are saved into the new file.

Cancel - Will discard all changes and close the dialog.

6.6 General Operations

This section describes JRunner operations that are common to the JRunner module and to the scenario studio module.

6.6.1 Switch Project

In order to change the automation project that appears in the JRunner, select the **"Switch Project"** option from the **"File"** menu.

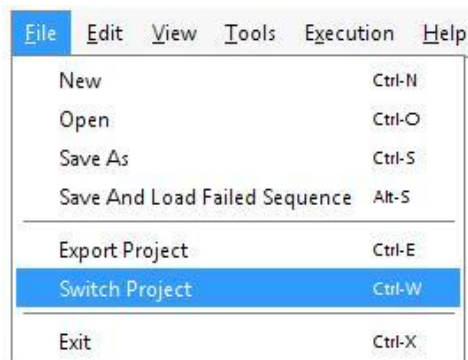


Figure 48: Switching Projects

Once the selection has been made, the **"Select tests classes directory"** dialogue opens, browse to the classes folder of the automation project that you require and select it.

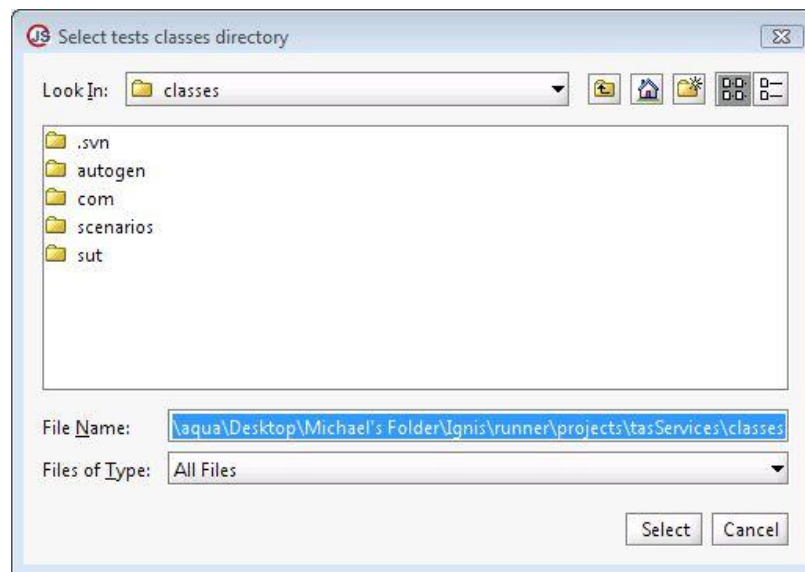


Figure 49: Selecting Test Class Directory

Note: The JRunner validates that the selected automation project includes the "SUT" package in the "tests" folder and in it at least one SUT file. If not, the user is asked to re-select the automation project folder.

6.6.1.1 Refresh

After making code changes or configuration changes (by altering the jsystem.properties file), new compiled code and or configuration have to be reloaded into the JRunner. In order to reload configurations and or code press on the 💰 "Refresh" button.



Figure 50: Refreshing a Project

6.6.2 Export Wizard

The purpose of the export wizard is to allow the user to pack and compress the automation environment after it is debugged and tested, now deploy the packed environment on a clean machine.

In order to export the automation environment select the "Export Project" option from the "File" menu, or press on the 📦 "Export Project" tool bar button.

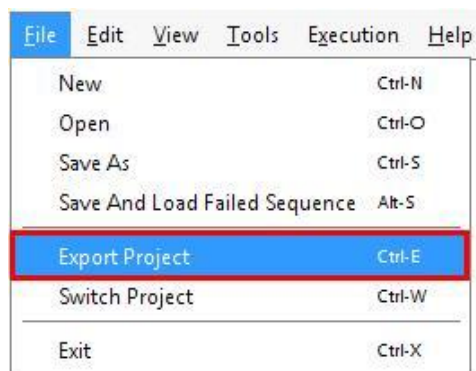


Figure 51: JSystem Report Wizard

Pressing on the Export Project button initiates a wizard. Now press the "next" button in order to configure the export information.

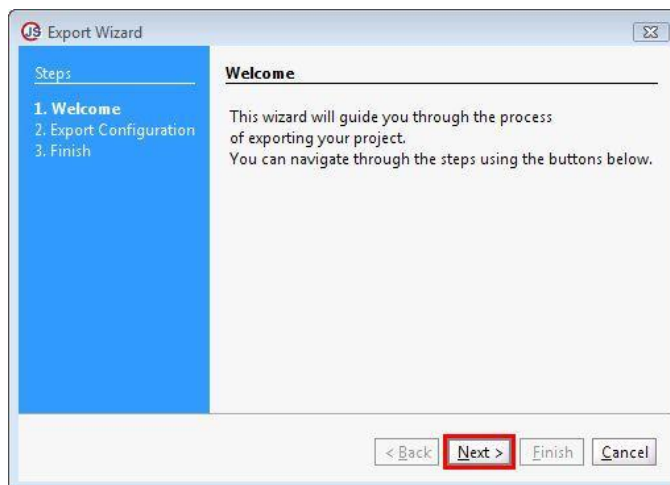


Figure 52: First Export Wizard Window

6.6.2.1 Export Configuration

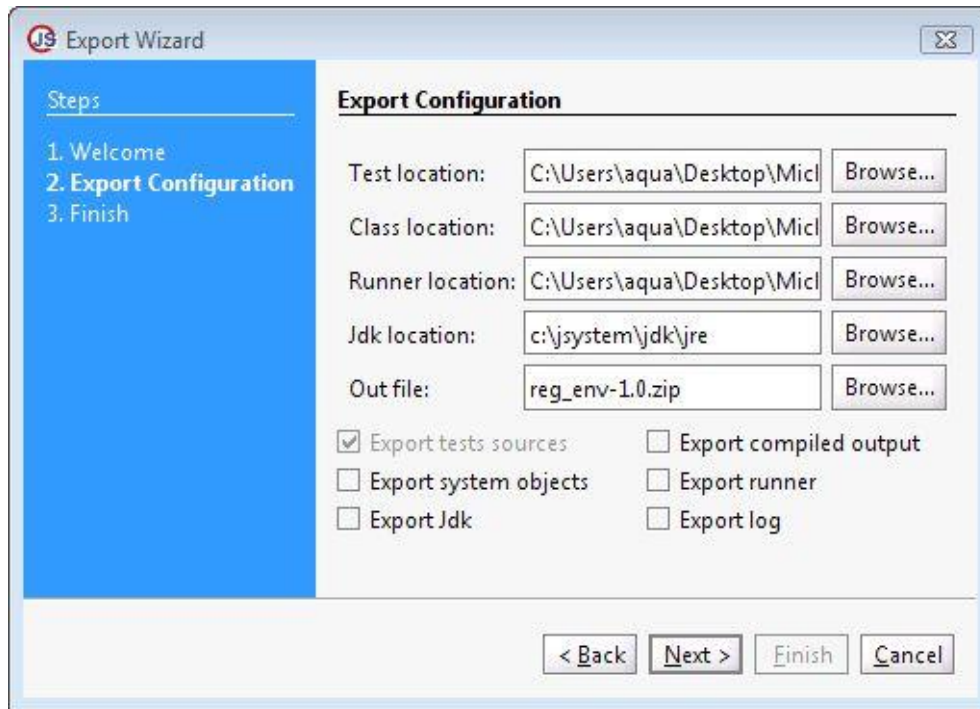


Figure 53: Export Configuration Wizard

By default the export wizard packs the automation project code that can be extracted on the destination machine and compiles it on the destination machine. The export wizard takes the test sources and class paths from the "**jssystem.properties**" file. In order to pack a project in a different directory path, change the "**Test location**" field.

In addition to automation project code the user can pack the following elements:

- **Export JRunner** - When selecting the "**Export JRunner**" check the JRunner packs in the export zip file. In this configuration, the destination machine does not have to assume that JRunner is installed.
- **Export Log** - When selecting this option, the internal JRunner log files are packed. This option is used for support calls.
- **Export Compiled Output** – Trigger additions of compiled code into an export package; this option is used if the destination machine does not have a java development environment installed and the source classes cannot be compiled on it.
- **Export JDK** - When selecting the "**Export JDK**" checkbox, the JRunner packs in the export zip file the JRE that the JRunner works with. This option provides that the destination machine does not have to assume that JRE is installed on it.

Note: The path of the packed environment can be controlled by altering the "Out file" field.

If a folder with the name "resources" exists in the automation project the resources folder should be a sibling of the "classes" folder, it is compressed in the zip file that is created by the export wizard.