Version 5.0

Release Notes

Disclaimer

Table of Contents

Table of Figures

List of Tables

1.1 The Purpose of this Document

The release notes describe the new features and bug fixes for the JSystem release version 5.0 of the JSystem framework.

1.1.1 JSystem Features Summary

- JSystem 5.0 includes features that span over all of the automation development lifecycle and users functionality. The new features include the following: Eclipse development plug-in. The plug-in greatly enhances developer capabilities enabling a feature that makes it easy to create a JSystem development workspace.
- New development capabilities for users of the scenario studio (JRunner) with the scenario parameters and parameters reference feature.
- The ability to delete runs log from the JReports server, thus enabling large organizations to use the server.

1.2 Eclipse Environment Settings

JSystem 5.0 comes with an Eclipse plug-in. The first version of the eclipse plug-in is designed for java developers, helping create automation workspaces in the eclipse development environment according to JSystem methodology conventions.

1.2.1 Plug-in Installation

In order to install the JSystem plug-in perform the following steps:

1. In eclipse, select Help→Software Updates→Find and Install

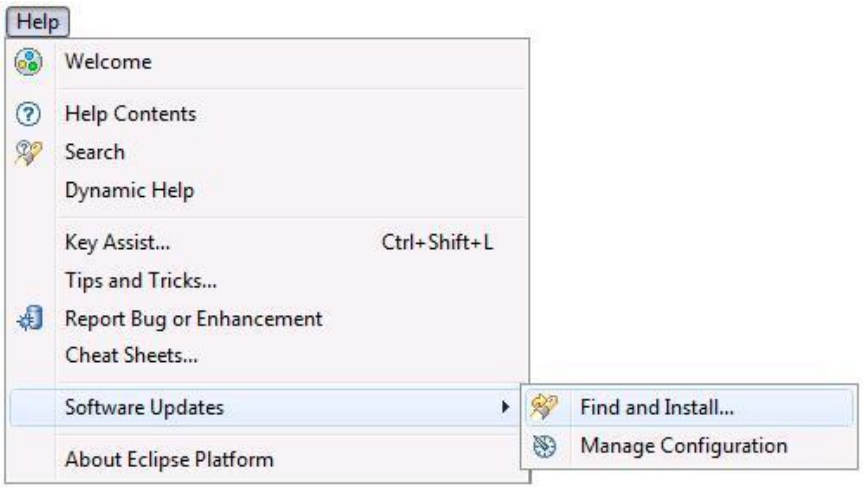


Figure 1: Java Eclipse SDK

2. Select "**Search for new features to install**" radio button.

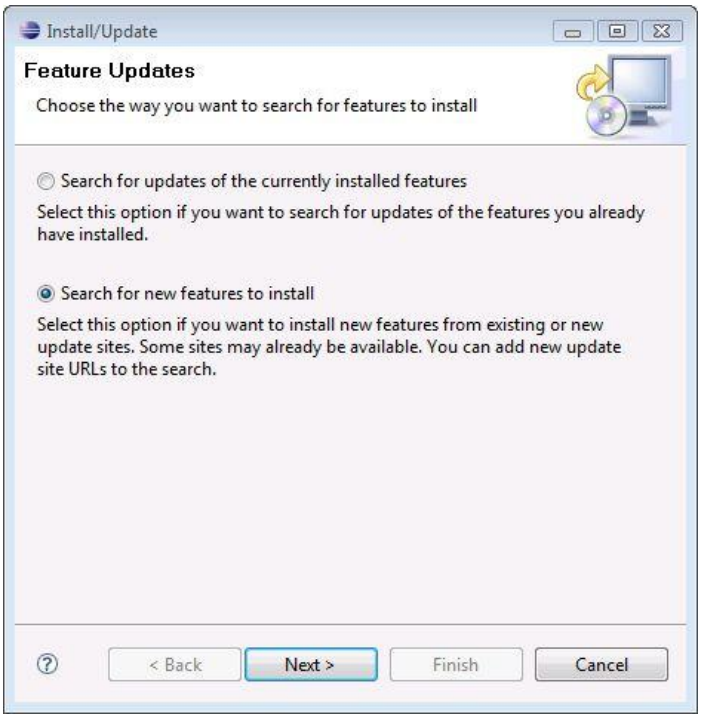


Figure 2: Install Updates Panel

3. Now click on the "**New Remote Site**" button.

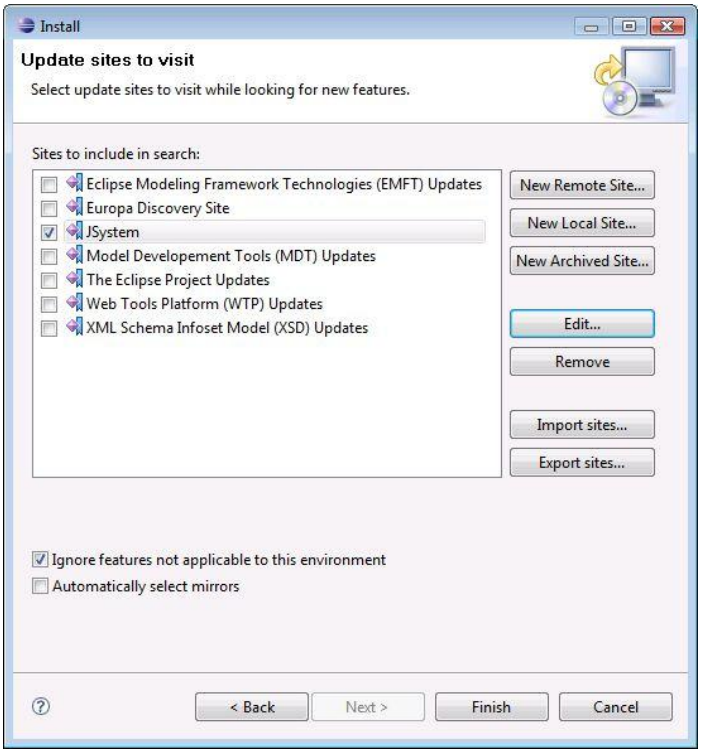


Figure 3: Install Pane

1. Enter the following URL: http://ignis.deepdev.net/update.

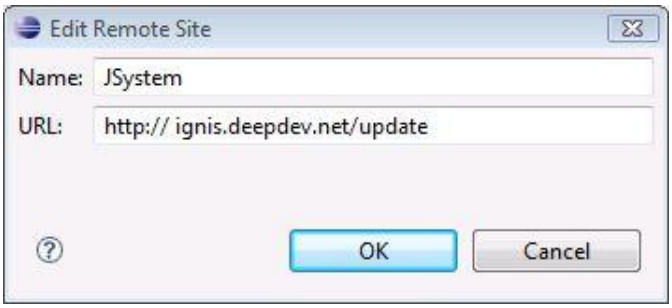


Figure 4: Edit Remote Site

2. Make sure only JSystem is selected in plug-ins list.

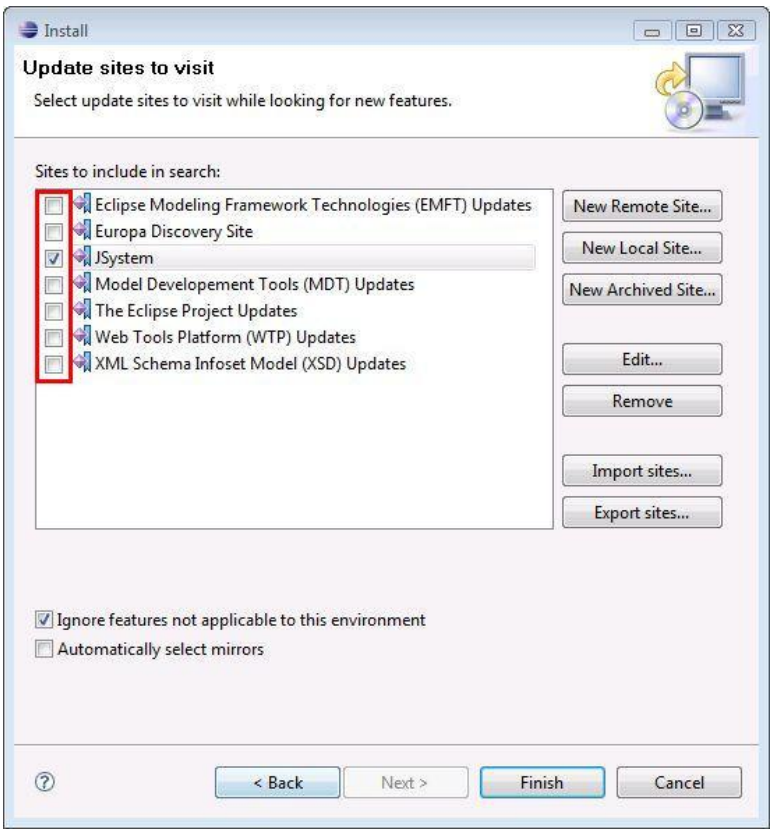


Figure 5: Update Site to Visit

3. Click on the "**Finish**" button.

4. Select the JSystem entry and click the "**next**" button.

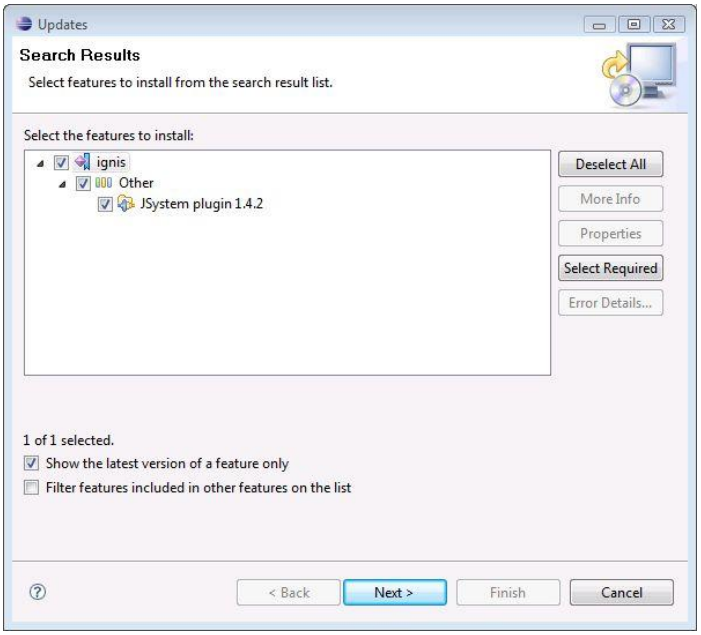


Figure 6: Update Search Results

5. Now, accept the license agreement and click on the "**next**" button.
6. Click on the "**Finish**" button.

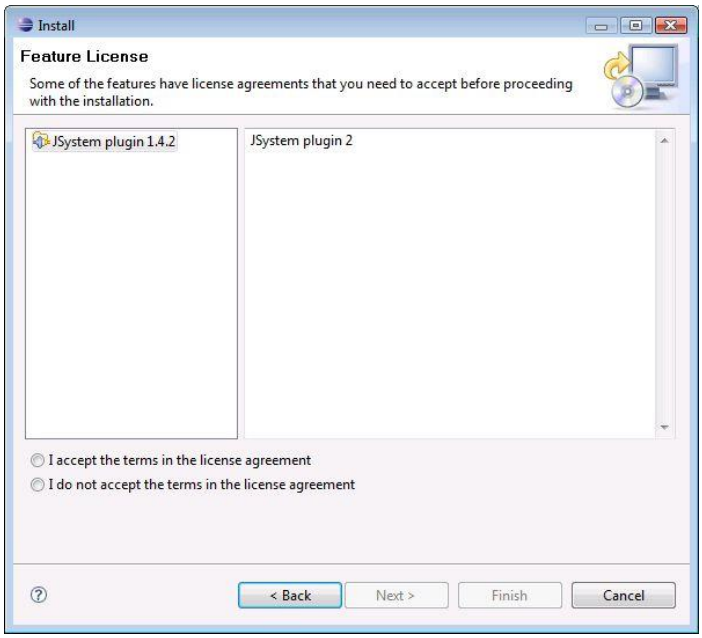
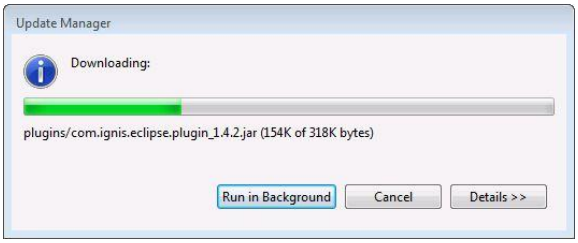


Figure 7: Installation Pane



4. Click on the "**Install All**" button.

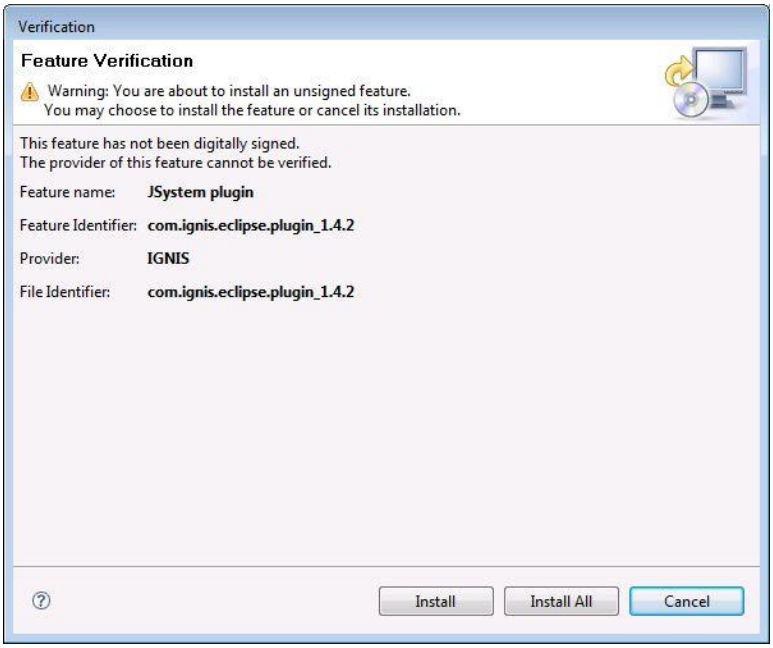


Figure 8: Feature Verification

7. Once installation has completed, the restart prompt is shown now click on the "**Yes**" button.

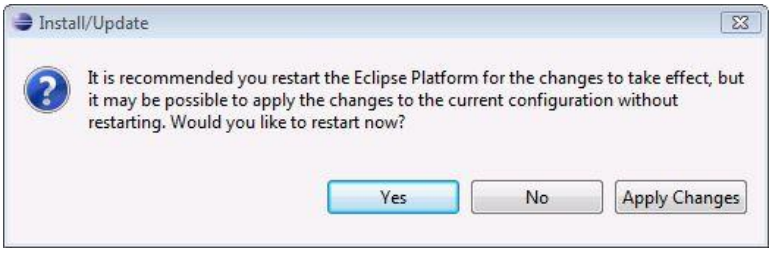


Figure 9: Install Update Window

1.2.2 Using the Plug-in

The plug-in enables the developer the ability to create a development workspace. The stages in creating a workspace are importing the JRunner into the workspace and then creating system objects and tests projects.

1.2.2.1 Importing JRunner

To import the JRunner into the workspace perform the following steps.

1. Select "**Preferences**" from the Window menu.

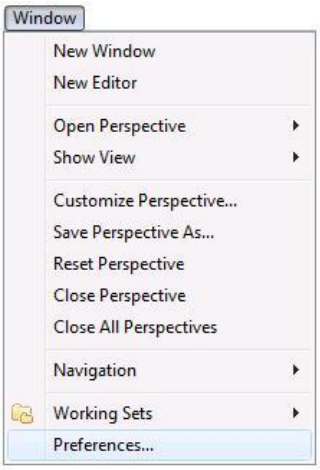


Figure 10: Eclipse SDK New Window

2. Select the "**JSystem Preferences**" option.

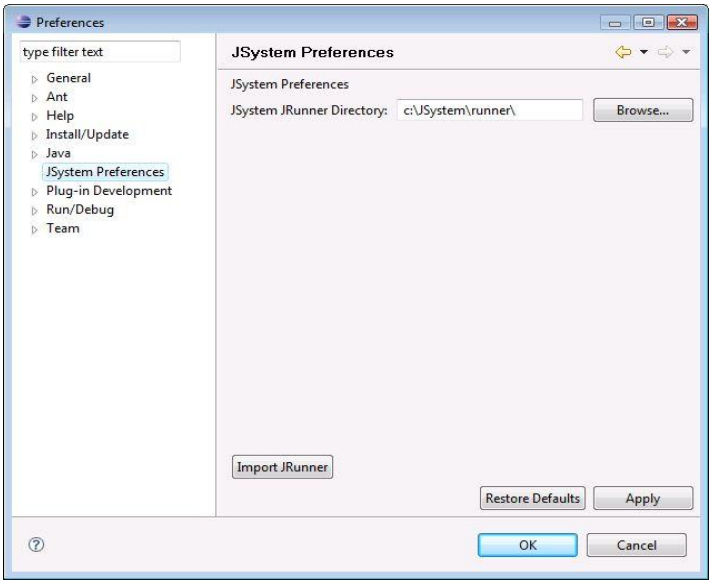


Figure 11: JSystem Preferences

3. Enter the installation path of the JRunner in JSystem runner directory.
4. Click on the "**Import JRunner**" button.

1.2.3 Creating a System Object Project

To create a new system object project do the following:

1. Select "**New"** and then select "**Project**" from the File menu.

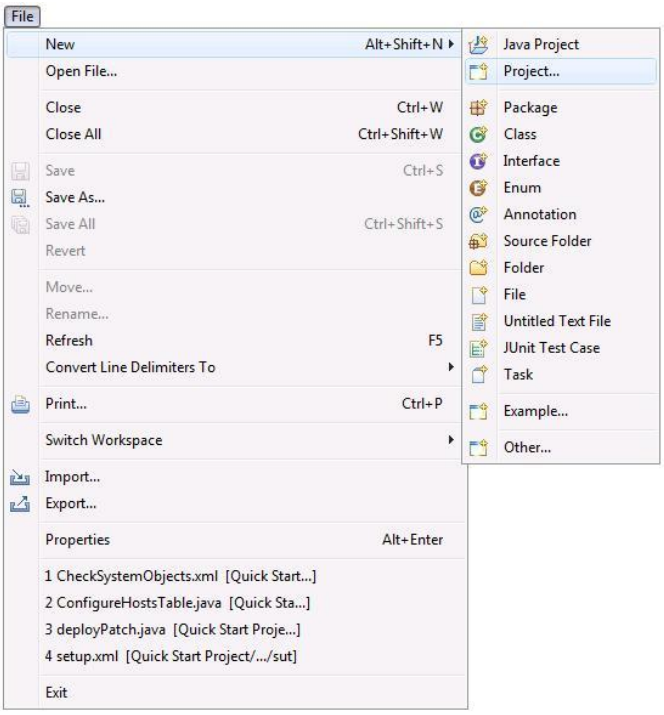


Figure 12: New Eclipse Project

2. Expand the JSystem folder and choose the "**JSystem System Object project**" option in the "**New Project**" dialog.
3.

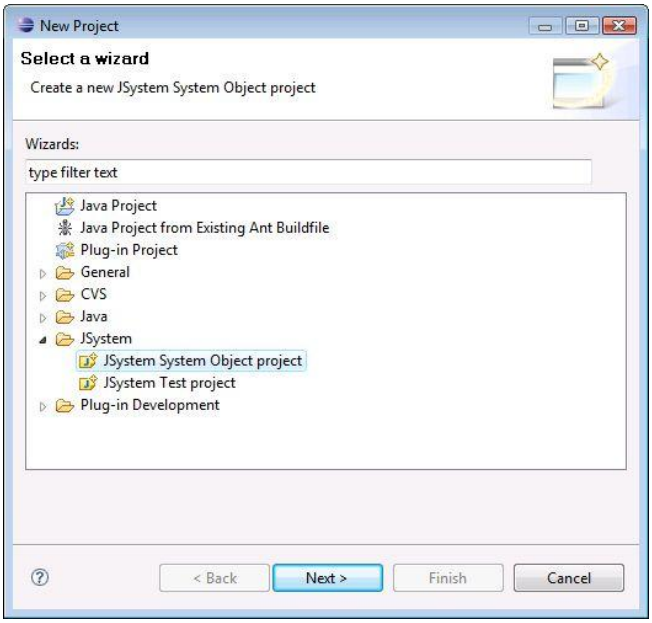


Figure 13: New System Object Project

4. In the New project dialog, name your system object project and select the drivers that are required for use in the system object

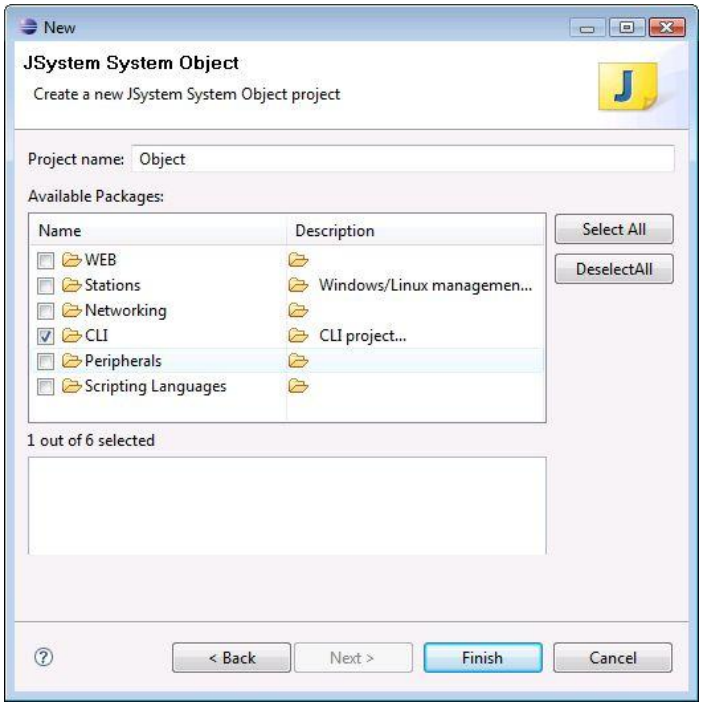


Figure 14: System Object Wizard Completion

5. After clicking the "**Finish**" button a "**system object**" with example code and build.xml file are created.

1.2.4 Creating a Tests Project

In order to create a new system object project perform the following steps.

1. Form the "**File**" menu select "**New"** and then select "**Project**".

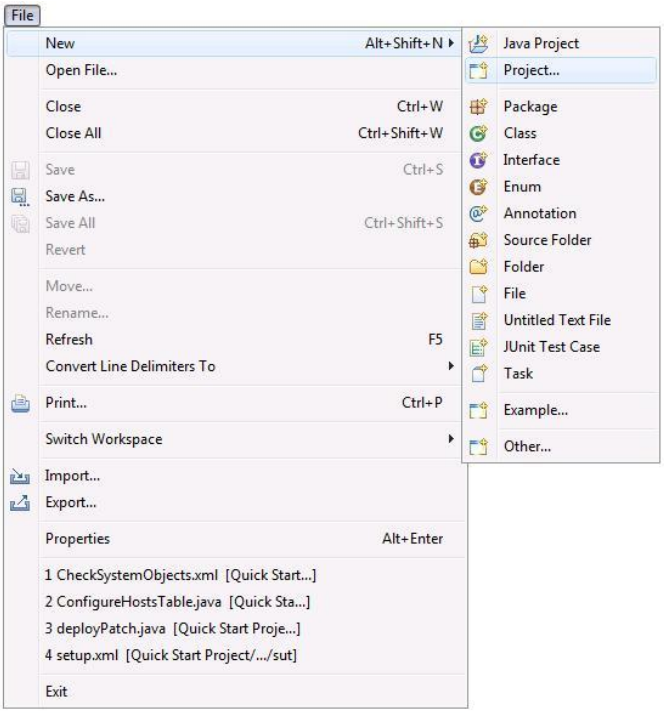


Figure 15: Creating a New Test Project

2. Once the "New Project" dialog opens, choose the JSystem folder and then select the "**JSystem Test project**", and click "**Next**".

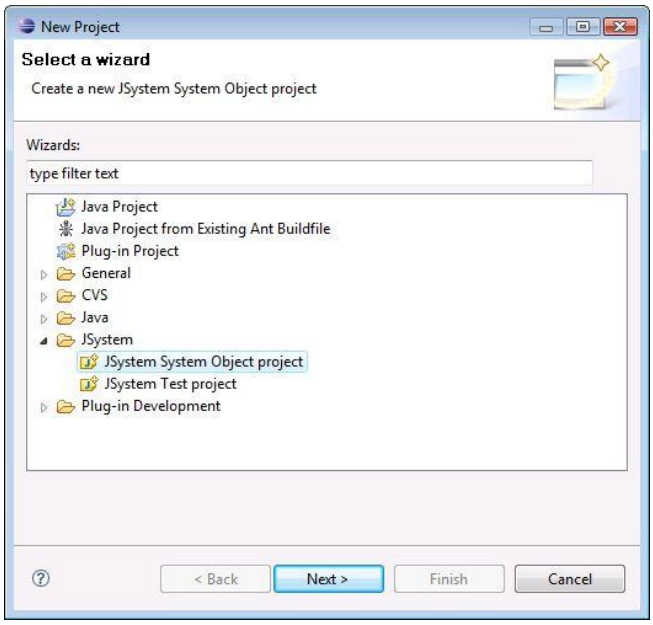


Figure 16: JSystem New Project Wizard

3. In the New project dialog, give the tests project a name and select the drivers that are required by the system object.

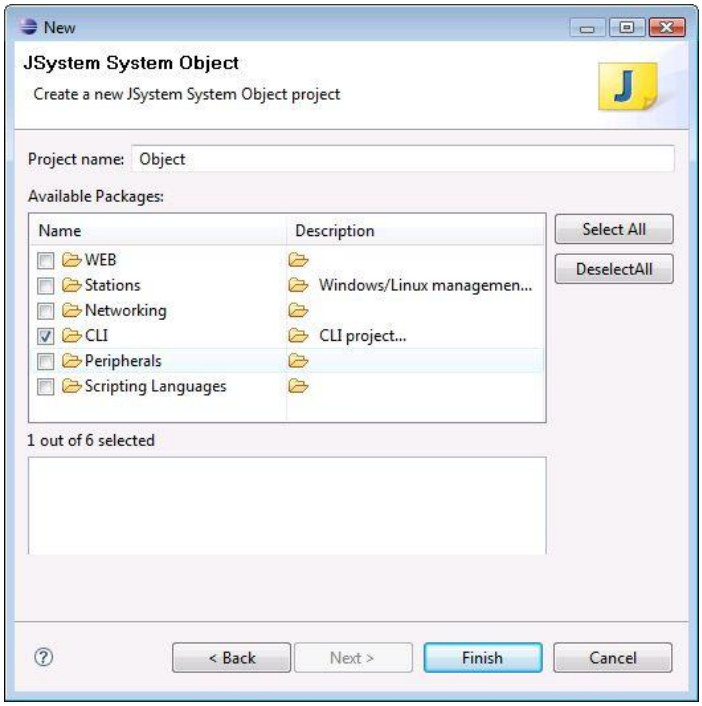


Figure 17: JSystem New Project Wizard 2

4. After selecting all of the relevant drivers click the "**Finish**" button. A test project with all relevant folders is created.

1.3 JRunner Enhancements

The following section describes the latest features and enhancements in the JSystem JRunner 5.0 release.

1.3.1 JSystem Agent

JSystem 5.0 represents the first installment in the conversion of JSystem from a desktop application into a full client server application.

Alterations have been made to the JSystem JRunner in order to enable the first step in separating the execution engine and JRunner UI.

1.3.1.1 Installing and Running the JSystem Agent

No changes have been made to the JSystem installation. At the end of the installation process there are four visible batch files:

- **run.bat (run for Linux)** – activates the JSystem JRunner UI. When activating the JRunner UI, the JRunner work as it always worked, it can also connect to a remote agent.
- **runAgent.bat (runAgent for Linux)** – activates runner agent. Runner agent is a console application (no UI). Agent can be activated/ controlled by runner UI or by native java JConsole application.
- **runBase.bat (runBase for Linux)** – base script file which is used by run and runAgent.
- **InstallJSystemAgent-NT.bat** – script which installs runner agent as a service on a windows machine.
  Issues to note when working with JSystem's JAgent:
  - In order to install the service on Windows Vista the script has to be activated by system administrator.
  - In order for the service to work properly, the JAVA_HOME environment variable must be defined.
  - Activating the JRunner agent as service on Linux is presently not supported.

Note: If you plan to work with the JRunner agent, make sure that the installation path does not have spaces in it (for example, don't install JSystem on c:\Program Files\...).

1.3.1.2 Configuring the JSystem Agent

The JSystem agent supports all JSystem properties that are related to test executions for example, run.mode, test.vm.params as all the previous versions have supported. These properties can be altered by changing the "**jsystem.properties**" file.

Additional parameters that are relevant to the agent:

- **RMI port** – The RMI is the main protocol used for connecting and activating the agent. The RMI server port can be configured in the runAgent script file or in the "**wrapper.conf**" file for the agent service. The default port is 8999.
- **Web port –** The JRunner agent has an embedded tomcat servlet that enables accessing HTML reports remotely. The HTTP port can be configured by setting the property "**agent.server.web.port**" in the "**jsystem.properties**" file. The default port is 8383.
- **FTP port** – The JRunner agent has an FTP server embedded in it that enables the transfer of automation projects from remote clients to the agent. The FTP port is configured by updating the property "**agent.server.ftp.port**" in the "**jsystem.properties**" file. The default port is 2121.
-

1.3.1.3 Working with the JSystem Agent

- First, run the agent on machine "a", either by activating the "**runAgent**" script or by registering the service and starting it.
- Now, run the JRunner UI on another machine by activating the run script.

Note: The agent and the UI can be activated on the same machine, but they need different installations.

- Use the JRunner in the standard manner, select a project, create and or edit the scenario, and then run the scenario and making sure that it is stable.
- Once you are performed the previous steps in the JRunner, select the "**View→Toolbars→Agent**" Toolbar.

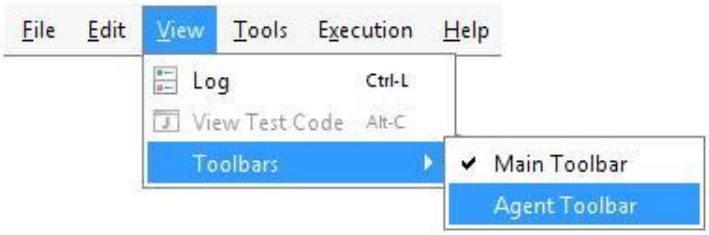


Figure 18: Setting the JSystem Agent

- The Agent toolbar appears.

Figure 19: JSystem Agent Tools

- **Agent List** - shows a list of agents that the GUI was recently connected to. The first agent in the list is the last agent that the runner was connected to. By default the JRunner starts with the "**local**" agent selected in the agents list. Local refers to the JRunner operates normally.
- **Connect -** Signals the GUI to connect to the agent currently selected in the agents list. The agent list combo box is editable, in order to connect to a new agent write "**agent_ip:rmi_port**" and press on the connect button.
- **Status -** shows the connection status: - Green = all is well, Red = Problems with agent connection, Grey – not connected

When the connection to agent is lost, the UI starts to monitor the connection. Once the agent can be reached the status button becomes green again.

- When the JRunner UI is connected to the agent the user can either continue to configure the scenario or press on play button.
  When pressing on the play button the following happens:
  - The UI signals the agent to switch to the automation project that is currently active in the client. The project is identified by the name of classes folder parent.
  - The JRunner asks the user whether to synchronize the agent with the automation project, if "**yes**" is selected the automation project is copied from the client to the agent (using the FTP server on the agent).

Note: In order to initiate the first run press the "yes" button to synchronize.

To implement a change in the scenario, press the "**yes**" button in order to synchronize.

Automation content that is transferred to the agent includes the following:

- Classes folder (includes compiled tests, scenarios and SUT files).
- "**lib**" folder (includes system objects jars).
- "**resources**" folder (includes additional resources that are needed by the automation project).
  - The UI signals the agent to switch to the currently active scenario and currently active SUT file
  - The JRunner takes relevant "**jsystem.properties**" on the client side and updates the agent with the props values.
  - The agent is signaled to run scenario.
  - Currently, the execution cannot be stopped while the agent is synchronizing with the client. The reporter tab is updated with remote activation progress.
- Once the execution has started, stop, graceful stop, pause, repeat work as usual.
- The "**reporter**" tab is updated as usual.

- When pressing on the "**log**" button, the browser is opened with a URL that points to the "**index.html**" on the remote agent.
- The following functionalities do not work when working on remote agent:
  - The jar list shows a list on the client and not on the remote agent.
  - Publishes through the UI, and works only on local agent.
  - Database properties updated manually on agent.

1.3.2 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>}**

1.3.2.1 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 your test and add it to your scenario.
3. Open the parameters panel of your test.

4. Add a reference.

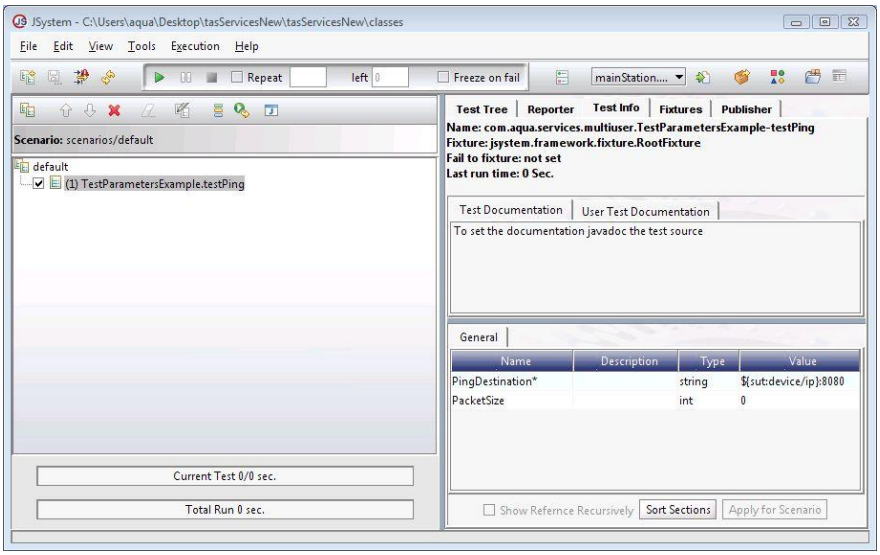


Figure 20: Adding a Reference

5. Zoom in:

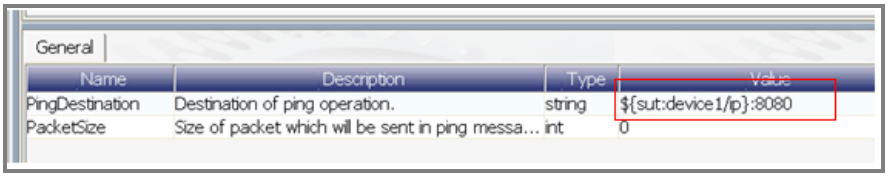


Figure 21: JSystem General Values

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.

1.3.2.2 RunProperties and Summary:

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

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

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

1.3.2.2.1 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 1: Run Properties Code Example

1.3.2.3 SUT File:

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

1.3.2.3.1 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 2: SUT Code Example

1.3.2.3.2 JRunner Parameter Values

${sut:device1/password}

${sut:obj}

1.3.2.4 Randomization

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

   Example

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

   Table 3: Randomization Code Example

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

   Example

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

1.3.3 Scenario Parameters

Starting at JSystem 5.0 the user of the JRunner can define scenario parameters similar to test parameters. This feature gives JRunner users the ability to build reusable scenarios, and make the JRunner a tool for advanced users that are not programmers.

1.3.3.1 Using Scenario Parameters

In the following example we will see a scenario which validates 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.

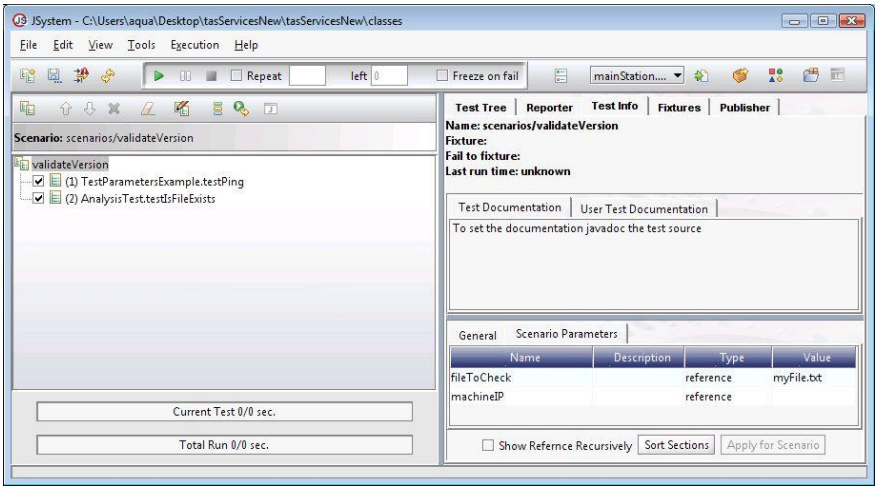


Figure 22: validateVersion scenario

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.

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

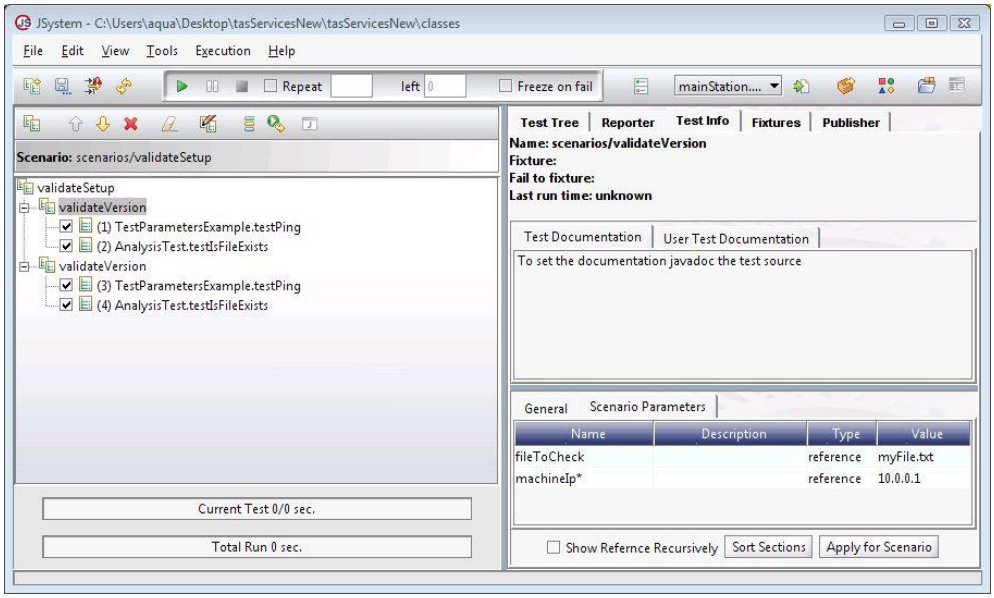


Figure 23: validateSetup scenario,

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.

In our example, set the "**machineIP**" of one instance to 10.0.0.1 and the second "**machineIP**" instance to 10.0.0.2.

After setting the scenario parameter value apply the operation by pressing on the "**Apply for Scenario**" button.

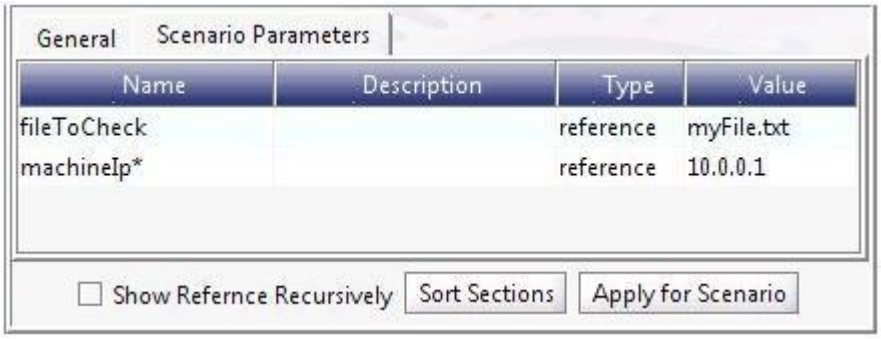


Figure 24: Applying for Scenario

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.

1.3.3.2 Defining Scenario Parameters

Now that we have seen how to use scenario parameters, let's see how we define scenario parameter:

1. Select the scenario for which you wish to define a parameter.
2. Assume that the name of the parameter being added is the "**machineIP**". Go to the applicable test that you want to use the scenario parameter and incorporate the scenario parameter in test parameter value in this format **${scenario:ParamName}.**

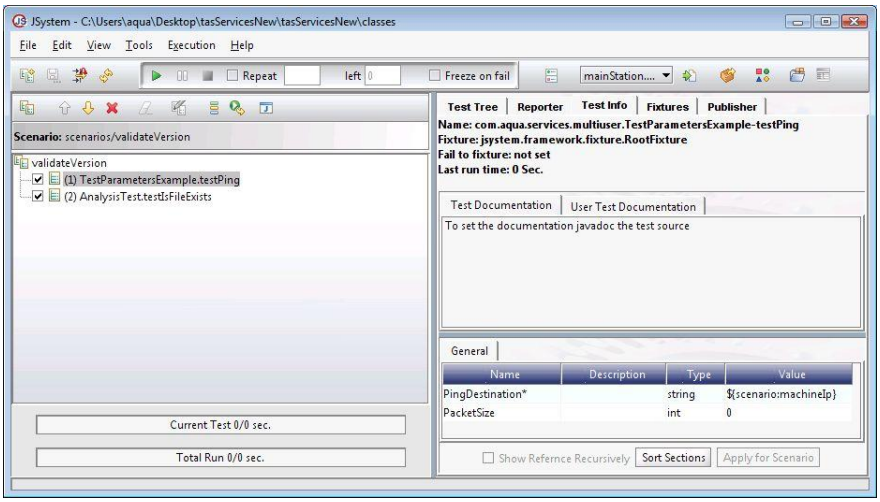


Figure 25: Defining Parameters

3. Zoom in.

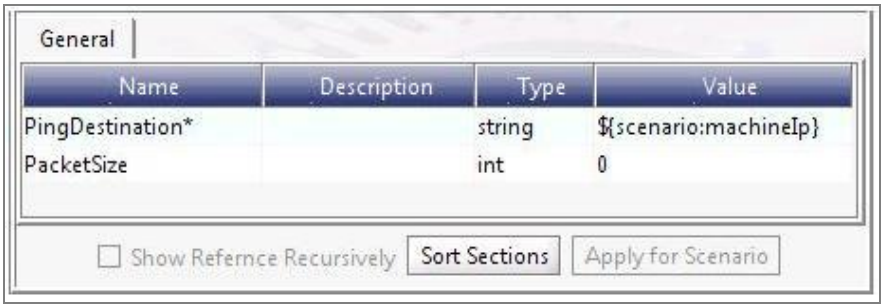


Figure 26: Setting Values

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

1.3.3.3 Scenarios Parameters with Nested Scenarios

How scenario parameters behave when the user nests several scenarios.

Take the "**validateSetup**" scenario and add it to a scenario called "**setupMaster**"

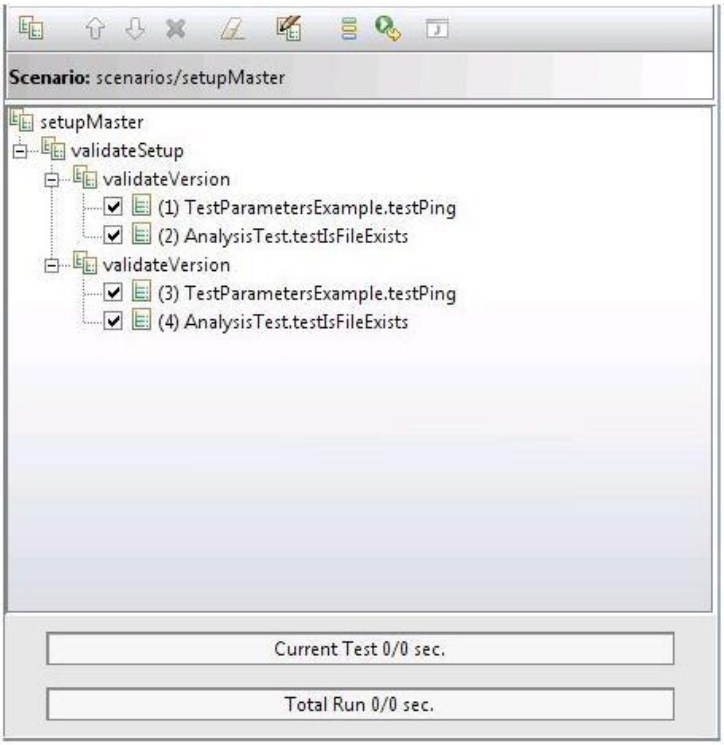


Figure 27: Nested Scenario Parameters

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.

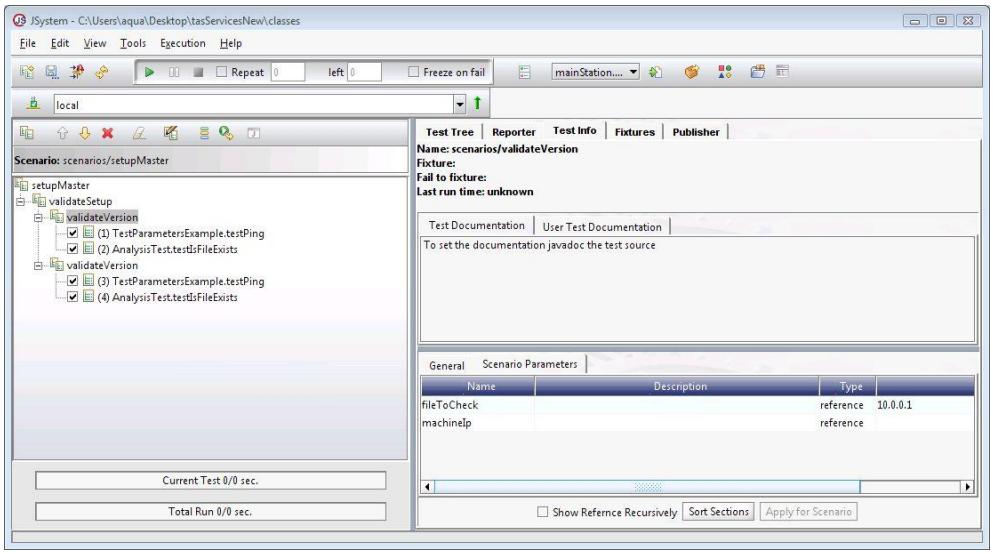


Figure 28: Validate Setup

Zoom in:

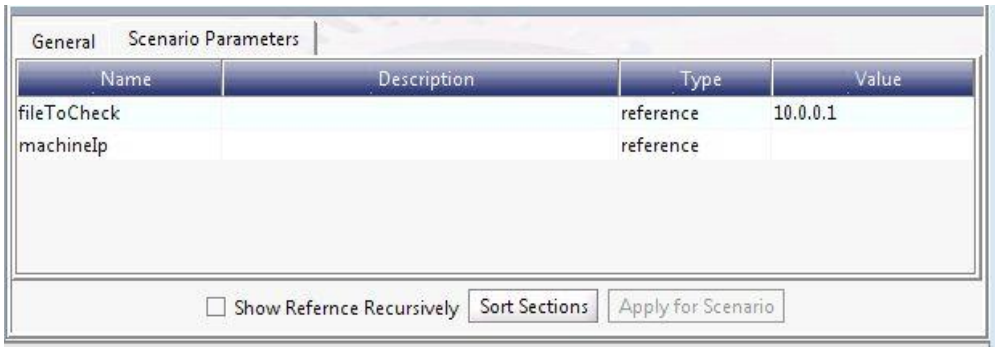


Figure 29: Scenario Parameters

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

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.

1.3.3.4 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 you will now select one of the instances of the "**validateVersion**" scenarios you will see that the values of the parameters were resorted to their default values.

1.3.3.5 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 30: Show Reference

1.3.4 Publish Event

The publish event was extended to allow two more features:

1. Gathering "**publish filter**" information from property files.
2. Mail notification.

1.3.4.1 Gathering Publish Filter Information from Property Files

Version, Build, and Description values can be gathered from either summary properties, or options file. Summary properties can be altered during scenario execution whereas options file is a static file.

1.3.4.1.1 Summary properties

In order to add a property to summary properties file add **"Summary.getInstance().setProperty(key,value)"** to a test code, when key can be "**Version**","**Description**","**build**".

For example: Summary.getInstance().setProperty("Version","1.5").

1.3.4.1.2 Options File

A file called "**publishEventOptions.properties**" can be placed in the JRunner's main folder in order to allow the user to define a group of values from which the publish event values will be selected. The values are semicolon separated.

For example: version=2;3;4;5.5

Note: If values are defined, a combo box appears in the JRunner allowing the user to choose from the parameters in the panel.

1.3.4.2 Mail Notification

The publish event was extended in order to allow mail notification of runs. The user can now receive an email with the run results and a matching link to the reports system.

In order to use this feature do the following:

1. Add the following keys to the jsystem.properties file:
   - **mail.host** - The SNMP server host address (example: Gmail – smtp.gmail.com)
   - **mail.port -** The server Port (example: Gmail – 465)
   - **mail.ssl -** Does the server require secured connection? (Gmail – true)
   - **mail.from** – The mail address that should appear as the sender.
   - **mail.user** - The account user name.
   - **mail.password** – The account password. (Encrypted after first time used).
   - **mail.sendTo** - **<optional>**- a semicolon separates a list of all the accounts it is sent to.
2. Add the "**PublishTest**" event to your scenario.

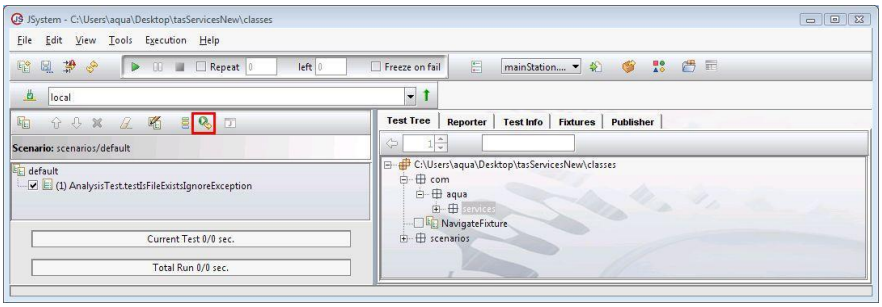


Figure 31: Publishing a Test

3. After "**PublishTest**" is added, double click on it and select the "**General**" tab in the parameters panel.

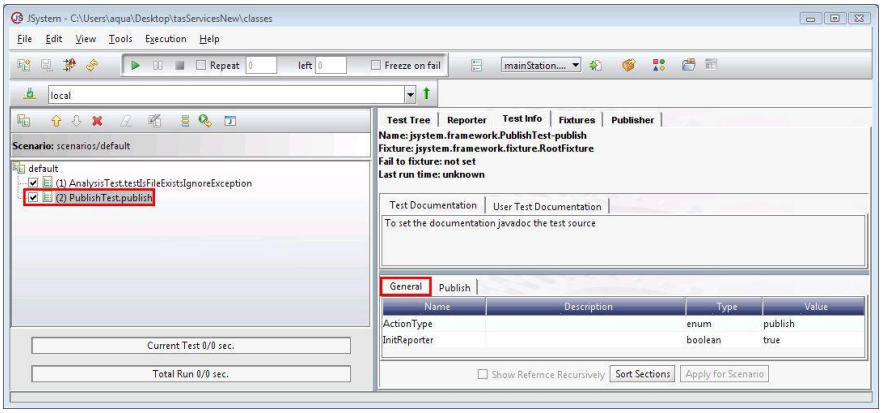


Figure 32: JSystem Parameters Panel

4. Change the value of action type to "**email**" or "**publish_and_email**":

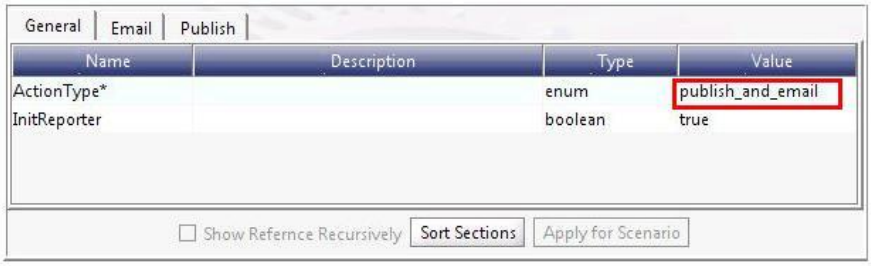


Figure 33: Changing Action Values

5. When selecting to send an email the user can modify the following Parameters:

- **Mail Subject** – the title of mail that is sent.
- **Message Header** – the message that is displayed in the mail body, before the results section.
- **Sent To List (semicolon separated)** – if any list was configured in the "**jsystem.properties"** it appears, the list is editable.

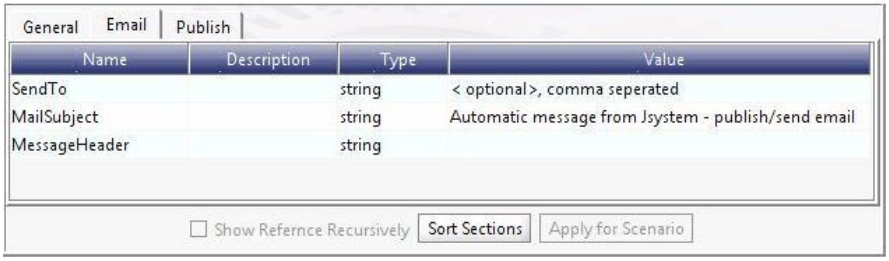


Figure 34: View Action Values

1.3.5 Graceful Stop

Previous to the JSystem 5.0 when ever scenario execution was stopped by the user, the tests JVM was immediately disposed.

In JSystem 5.0 when the user presses on the "**stop**" button the framework performs a graceful stop: a test tear down method is invoked and the system objects are signaled to close themselves.

During the stop process a dialog is shown.

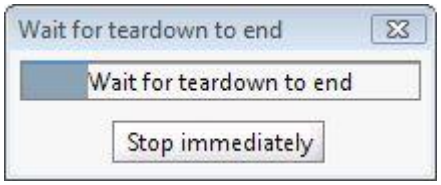


Figure 35: Teardown Notification

Pressing on the "**Stop immediately**" button causes the graceful process to terminate, and execution is stopped immediately.

1.3.6 Export Wizard Improvements

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.

1.4 JReporter Enhancements

Two feature enhancements have been added to the JReporter that enables using the JReport application in large organizations.

1.4.1 Run Filter Enhancement

The "**Version**" field was added to the runs filter:

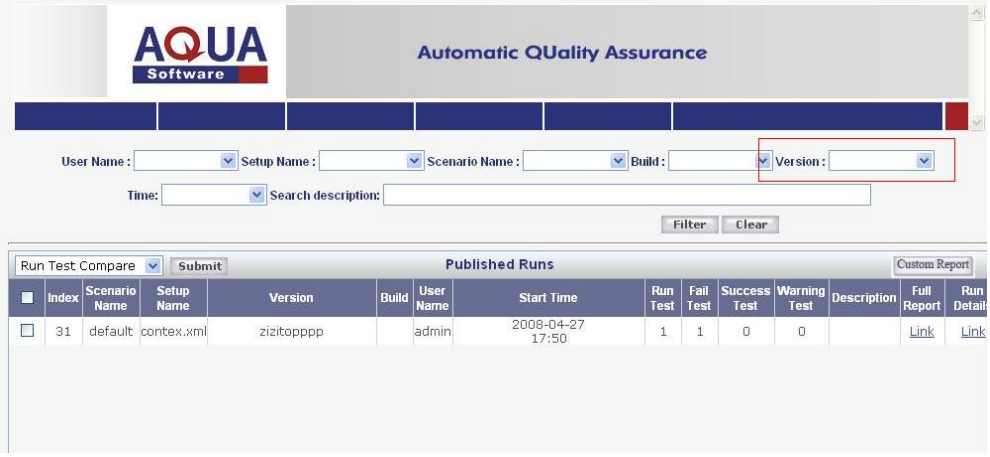


Figure 36: HTML JReporter

1.4.2 Runs Delete

Until JSystem 5.0 deletion of a run by the user , deleted run information from database but did not delete log files from file system. Since JSystem 5.0 when performing delete of runs, run log files are also deleted from file system.

1.5 API Enhancements

API enhancements where done to achieve the following:

1. Separation between JSystem components
2. Fix spelling mistakes and make API more self explanatory.

1.5.1 Jsystem.jar split

The JSystem jar has been split to four jars:

1. **jsystemCore** - Contains the core classes/apis of our system.
2. **jsystemCommon** - Common analyzers and utilities.
3. **jsystemAgent** - Separation of the runner engine from the runner application which knows how to execute scenarios/tests.
4. **jsystemApp** – The JRunner GUI and scenario editor.

From version 5.0 the JRunner lib will be comprised of four jars instead of the "**jsystem.jar**".
"**jsystemCore.jar, jsystemCommon.jar, jsystemAgent.jar"** and **"jsystemApp.jar".**

When creating a new system object project only the "**jsystem.core**" and the "**jsystemCommon.jar**" have to be added to the "**classpath**".

1.5.2 Methods Names Re factor

Different methods in different classes became deprecated to fix spelling mistakes and to make them more self explanatory.

1. **SystemObject interface** –" **testAgainsObject**" was deprecated and the method "**testAgainstObject**" has been added.
2. **Cli** - Merged the infra and "**telnetclient**" jars to Cli jar changed the method "**set/getScrallEnd**" to "**set/getDontWaitForScrollEnd**".
3. **Traffic** - in the "**TrafficPort**", the following methods deprecated

- public void "incrimentDestinationMac(int count)" throws an exception
- public void "incrimentSourceMac(int count)" throws an exception and added,
  - public void incrementDestinationMac(int count) throws Exception. public void incrementSourceMac(int count) throws Exception.