Version 5.1

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 5.1 of the ***JSystem Automation Platform***.

1.1.1 JSystem Features Summary

JSystem 5.1 includes features that span over all of the automation development lifecycle and users functionality. The new features include the following:

- Framework Improvements
- JSystem GUI Improvements
- Flow Control Enabled Scenarios
- Distributed Execution
- JUnit 4 Test Authoring Support
- Jython Test Authoring Support
- JSystem Properties Panel

1.2 Framework Improvements

Several new additions and improvements have been implemented into the JSystem framework in order to enhance the user experience and improve overall functionality.

1.2.1 Reporter

The reporting during a test can be set to a "**bold**" font in order to graphically show the difference between the specific report line and other reports. **usage: report.report("my title", ReportAttribute);** The **"ReportAttribute**" is an enumerator that can contain one the following values - BOLD, LINK, HTML, STEP.

1.2.2 Analysis

In addition to the standard pass and fail status that currently exists, a new "**Warning**" status also can be utilized helping differentiate between the test resulting in analysis failure. The analyze method has also been expanded to receive another Boolean parameter used to signal warning instead of a failure.

The New Method Signature

analyze(AnalyzerParameter parameter, boolean silent, boolean throwException, **boolean showAsWarning**).

1.2.3 Test and Parameters Annotations

Jsystem has extended the "**TestProperties"** annotation and added the "**ParameterProperties"** annotation. The annotations replace the old "**javadoc"** annotations used until the release of JSystem 5.1, such as "**params.include"**, "**params.exclude"** and section.

1.2.3.1 Annotations Usage

An annotation should be written just above the method it relates to, if there is also a "**javadoc"** for the method it should be between them.

For example,

```
/**
        * Test the parameters feature
        *
        */
       @TestProperties(name = "check that one parameter is
visible")
       public void testOneParamAndGibrish(){
              report.report("report something");
       }
```

Table 1: Annotations Code Example

Note: all annotation fields can be written in any order, comma separated.

Code Example

```
@TestProperties(name = "check that one parameter is visible",
        paramsInclude={"intValue"})
       public void testOneParamAndGibrish(){
              report.report("report something");
       }
```

Table 2: Annotations Code Example 2

1.2.4 Test Properties Annotation

TestProperties annotation is added before a test method and supports the following fields.

1. **name** – a meaningful test name, displayed on the Scenario tree and in the html logs. String.
2. **Group** - used for filtering tests in the JRunner and String array.
3. **returnParam** – used to define return parameters used by the flow control. String array.
4. **paramsInclude** – defines which test parameters are visible in the JRunner GUI for the test method. (An empty string means "**reveal no parameters**"). String array.
5. **paramsExclude** - defines which test parameters are not visible in the JRunner GUI for the test method. (An empty string means "r**eveal all parameters**"). String array.

Note: "paramsInclude" has priority and overrides "paramsExclude".

Note: The correct syntax for an empty parameters list is "paramsInclude={""}" and not "paramsInclude={}". The same is true for "paramsExclude".

1.2.4.1 Test Properties Annotation Code Example

The following code example refers to test property items four and five.

An example of a method in earlier JSystem versions.

```
/**
        * Test the parameters feature
        * @params.include intParam stringParam
        */
       public void testEmptyInclude(){
              report.report("intParam = "+intParam);
       }
```

Table 3: Old Code Example

An example of a method in JSystem version 5.1.

```
/**
        * Test the parameters feature
        */
       @TestProperties(paramsInclude = {"intParam","stringParam"})
       public void testEmptyInclude(){
              report.report("intParam = "+intParam);
       }
```

Table 4: New JSystem 5.1 Method Code Example

1.2.5 ParameterProperties Annotation

ParameterProperties annotation is added before a parameter setter and supports the following fields:

- **section** – define the tab this parameter will belong to in the runner parameters - panel. String
- **description** – a short description for the parameter, will appear in the parameters - panel

1.2.5.1 ParameterProperties Annotation Code Example

a setter code example as it appears in earlier JSystem versions.

```
/**
 * the user name for the DB
 * @section MySection
 */
public void setStringParam(String stringParam) {
      this.stringParam = stringParam;
}
```

Table 5: Parameter Properties Old Code Example

An example of a the parameter properties code example in JSystem version 5.1.

```
@ParameterProperties(section="MySection",
                     description=" the user name for the DB")
      public void setStringParam(String stringParam) {
            this.stringParam = stringParam;
      }
```

Table 6: New Parameters Property Code Example

1.3 JSystem GUI Improvements

In earlier releases of **JSystem Automation Platform**, the scenario tree expanded after every refresh, move and delete action.

1. Jsystem 5.1 now sports improved scenario tree functionality that includes the collapsibility and expandability of the paths saved.
2. An "**expand all**" and a "**collapse all**" option has also been added to the sub-scenarios. In order to access the menu, place the cursor over the test name in the scenario panel AND "**Right Click**" on it.

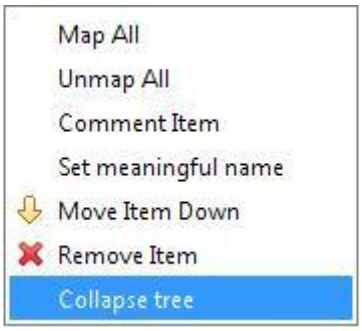


Figure 1: Collapse Tree Menu

1.3.1.1 Customer Product Property

JSystem now has the ability to add customer specific data to the JRunner "**About Window**" allowing the customer to present the current automation project version being worked on or any other requested info. In order to add Customer information to the "**About Window**", a JSystem property called "**customer.product**" must be set in the "**jsystem.properties"** file. The value is a semicolon separated string with different values, for example,
customer.product = "apcon-1.5;ixia 5.5;version=6" every semicolon separated value is displayed in the About window on a new line.

1.3.2 Publish Event Improvements

In addition to all prior publish event parameters, we have added three more parameters.

1. Scenario name - Default value is root scenario.
2. Sut - Default value is current sut.
3. Station - Default value is current station

All of the parameters are assigned default values when a publish event is added and they can be modified.

Important Note: The publish event package has been modified, so that when opening a scenario prior to Jsystem 5.1 that contains a publish event the user will receive an "error warning" and will need to remove and re-add the publish event to the scenario manually.

1.4 Flow Control Enabled Scenario

A new feature called "**Flow Control**" has been added to the JSystem Automation Platform. The flow control feature expands the capabilities of a test scenario to include a GUI based scripting mechanism and complex flow control, provides 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**".

1.4.1 Enabling Flow Control

The flow control feature is enabled by default. 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.

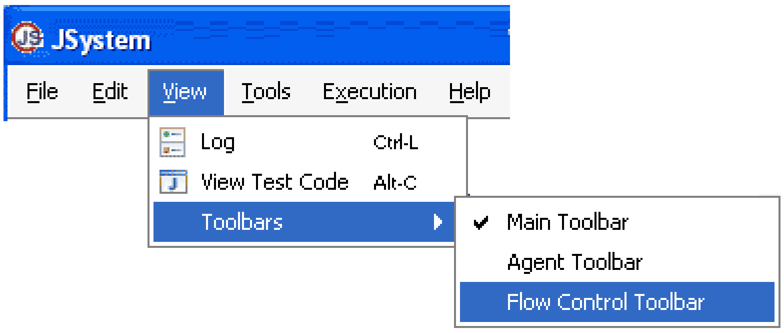


Figure 2: Flow Control Menu

1.4.2 Return Parameters

JSystem Automation Platform now 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.

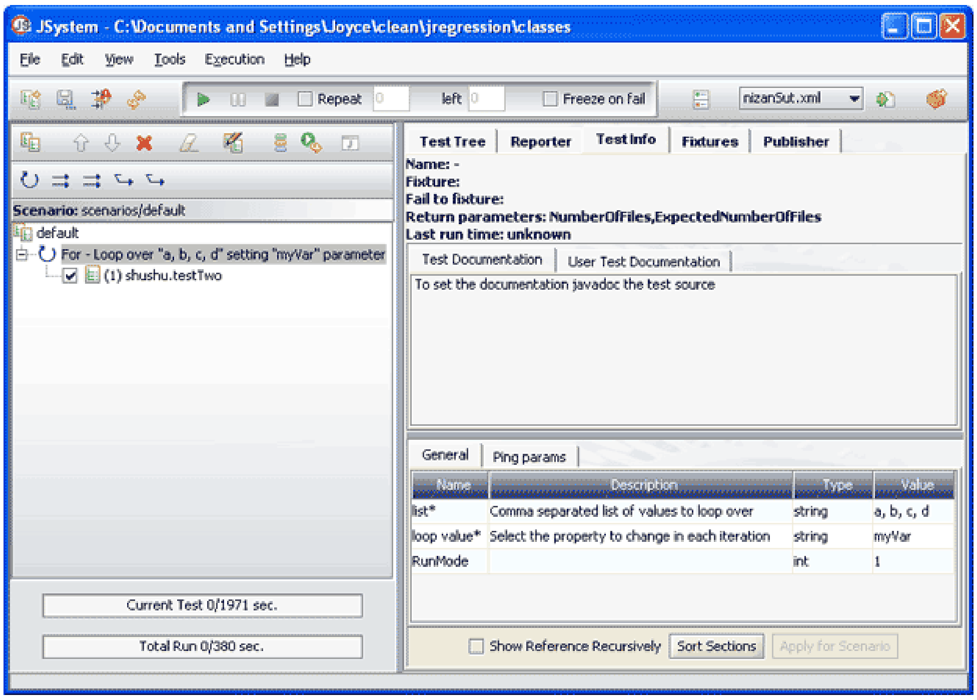
```
package com.aqua.sanity;

import jsystem.framework.TestProperties;
import com.aqua.general.JSysTestCase;

public class FlowControlFunctionality extends JSysTestCase {
    private int NumberOfFiles = 42;
    private int ExpectedNumberOfFiles = 3;

@TestProperties(returnParam={"ExpectedNumberOfFiles","NumberOfFiles"})
    public void testNumberOfFiles(){
        assertEquals(getNumberOfFiles(), getExpectedNumberOfFiles());
    }

    public int getNumberOfFiles() {
        return NumberOfFiles;
    }

    public int getExpectedNumberOfFiles() {
        return ExpectedNumberOfFiles;
    }

    public void setExpectedNumberOfFiles(int expectedNumberOfFiles) {
        ExpectedNumberOfFiles = expectedNumberOfFiles;
    }

    public void setNumberOfFiles(int numberOfFiles) {
        NumberOfFiles = numberOfFiles;
    }
}
```

Table 7: Flow Control Code Example

1.4.3 For Control

The "**for**" 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.



Scenario: scenarios/default

default
For - Loop over "a, b, c, d" setting "myVar" parameter
(1) shushu.testTwo

Current Test 0/1971 sec.

Total Run 0/380 sec.

Name: -
Fixture:
Fail to fixture:
Return parameters: NumberOfFiles,ExpectedNumberOfFiles
Last run time: unknown

To set the documentation javadoc the test source

Name	Description	Type	Value
list*	Comma separated list of values to loop over	string	a, b, c, d
loop value*	Select the property to change in each iteration	string	myVar
RunMode		int	1

Figure 3: Creating Flow Controls

The following changes and enhancements can be noticed by looking at the screen shot.

1. The new 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.

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 will be 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}.

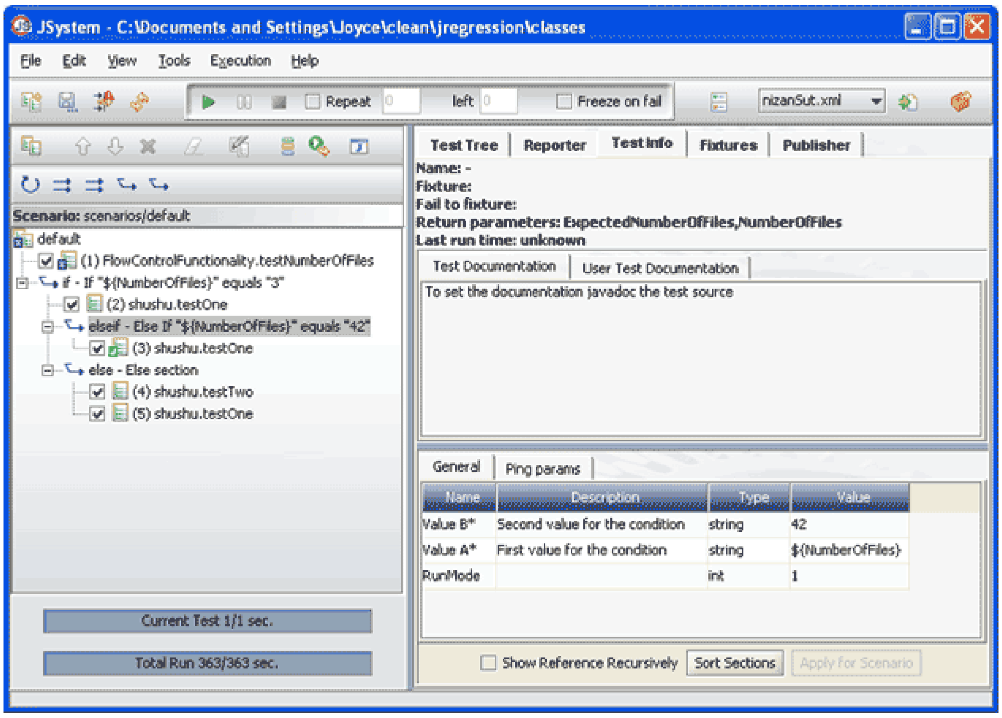
Note: In order to use the loop variable in tests inside the loop, set one of the test's parameter to be the loop variable. In the example, the value of a parameter in the loop is set to ${myVar}.

1.4.4 If Control

The "**if**" 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 a follows.



JSystem - C:\Documents and Settings\Joyce\clean\jregression\classes

Scenario: scenarios/default

```
default
  (1) FlowControlFunctionality.testNumberOfFiles
  if - If "${NumberOfFiles}" equals "3"
    (2) shushu.testOne
    elseif - Else If "${NumberOfFiles}" equals "42"
      (3) shushu.testOne
    else - Else section
      (4) shushu.testTwo
      (5) shushu.testOne
```

Current Test 1/1 sec.

Total Run 363/363 sec.

Name: -
Fixture:
Fail to fixture:
Return parameters: ExpectedNumberOfFiles,NumberOfFiles
Last run time: unknown

To set the documentation javadoc the test source

Name	Description	Type	Value
Value B*	Second value for the condition	string	42
Value A*	First value for the condition	string	${NumberOfFiles}
RunMode		int	1

Figure 4: Adding an If Flow Control

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

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

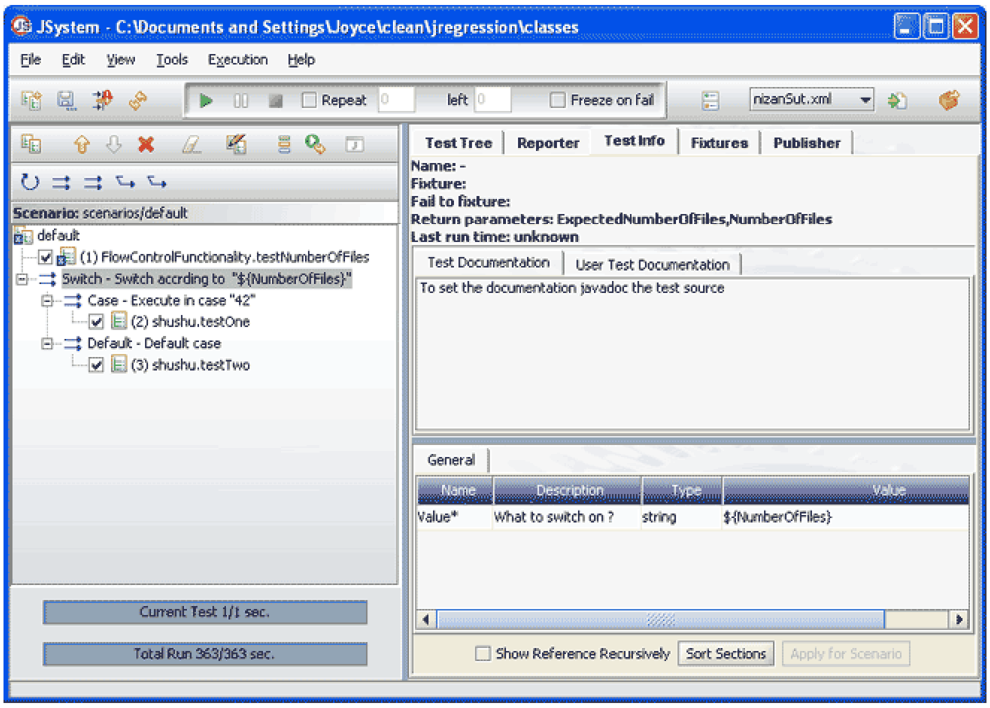
Note: Known limitation – The parameter name must use a capital Letter at the beginning of the word (Even when the actual parameter name uses a lower case character).

1.4.5 Switch Control

The "**switch**" 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 into 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 "**switch**" control to a scenario.



Scenario: scenarios/default

```
default
├─ (1) FlowControlFunctionality.testNumberOfFiles
└─ Switch - Switch accrding to "${NumberOfFiles}"
   ├─ Case - Execute in case "42"
   │  └─ (2) shushu.testOne
   └─ Default - Default case
      └─ (3) shushu.testTwo
```

Name: -
Fixture:
Fail to fixture:
Return parameters: ExpectedNumberOfFiles,NumberOfFiles
Last run time: unknown

To set the documentation javadoc the test source

Name	Description	Type	Value
Value*	What to switch on ?	string	${NumberOfFiles}

Figure 5: Adding a Switch Flow Control

1.4.6 Known Limitation

- No support for Excel import or export when using flow control.
- Scenarios that contain "**Flow Control**" elements can only run when configured to drop every run in "**Run Mode**".

1.5 Distributed Execution

The JSystem Automation Platform version 5.1 now supports a distributed execution of a JSystem scenario. The components that comprise a distributed setup are one or more JRunner agents and a JSystem execution station with JRunner.

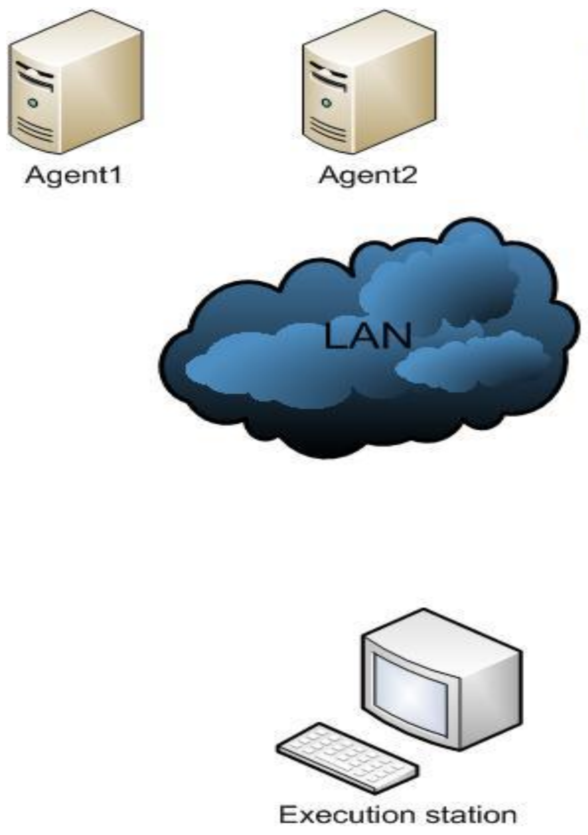


Figure 6: Distribution Execution Diagram

1.5.1 General Workflow of Distributed Environments

The following list details the basic steps required for working with this "**Agent Distribution**".

1. Install agents.
2. Add agents to the JRunner in the execution station.
3. Design the scenario.
4. Assign agents to sub-scenarios/tests.
5. Run the scenario.
6. Analyze the results.

Note: In order to get and handle events that come from the executing agent correctly, the version of the remote agent should be identical to the version of the JRunner in the execution station, in addition, all agents must be synchronized with the execution station automation project

1.5.2 JRunner Agent Setup and Installation

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

- **run.bat (run for Linux)** – activates the JSystem JRunner UI. The JRunner can also connect to a remote agent.
- **runAgent.bat (runAgent for Linux)** – activates the JRunner agent. The JRunner agent is a console application (no UI). The agent can be activated and or controlled by the JRunner UI or by a native java JConsole application.
- **InstallJSystemAgent-NT.bat** – the script that installs the JRunner agent as a service on a windows machine.

It is important to pay attention to the following settings and configurations when working with JSystem's JAgent:

- In order to install the service on the Windows Vista operating system 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 a Linux machine is currently not supported.

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

1.5.2.1 Configuration

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

Additional "agent parameters" that are relevant:

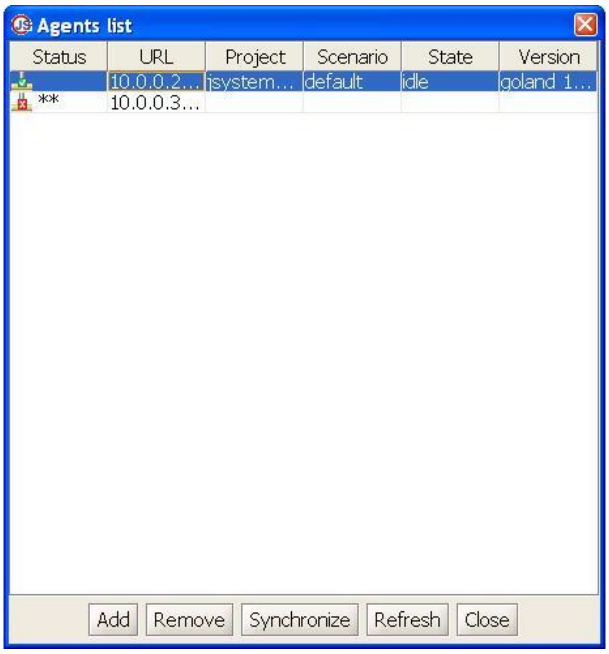
- **RMI Port** – The RMI is the main protocol used for connecting and activating the agent. The RMI server port can be configured in the runAgent script file or in the "**wrapper.conf**" file for the agent service. The default port is **8999**.
- **Web Port** – The JRunner agent has an embedded tomcat 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**.

1.5.3 Working with a Distributed Setup

Once agents are installed in the setup, the JRunner in the executing station has to be configured in order to recognize the agents; the user can then associate scenarios and or tests with one or more agents and then run the scenario.

1.5.4 Managing Agents

In order to manage agents in the JRunner, select the Tools pull down menu and select the "**Agents List**" menu item.



Status	URL	Project	Scenario	State	Version
	10.0.0.2…	jsystem…	default	idle	goland 1…
**	10.0.0.3…				

Figure 7: Agent List Window

Note: the meaning of the two asterisks (**) is that the agent is associated with one of the sub-scenarios/tests of the currently active root scenario.

1.5.4.1 Agents Table

The Agents list dialog is divided into the following columns.

Status- The connection status to the agent, when the icon is red it means that either the agent is down or there is a network problem. When an agent is marked with ** it means that one or more sub-scenarios and or tests in the currently selected root scenario are configured to run on the agent.

URL - The URL of the agent, the "**host:port**" the URL that is used by the system as a unique ID to identify the agent.

Project - The name of the automation project that is currently active in the agent.

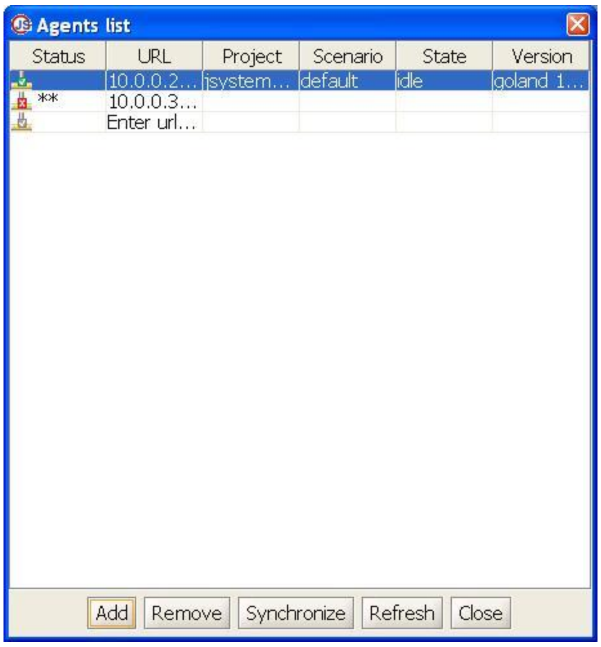
Scenario - The name of the root scenario in the agent.

State - Execution state (running/idle).

Version - Agent version.

1.5.4.2 Operations

- **Add** - By pressing the "**Add**" button a new line is added to the table.



Status	URL	Project	Scenario	State	Version
	10.0.0.2...	jsystem...	default	idle	goland 1...
**	10.0.0.3...				
	Enter url...				

Figure 8: Adding an Operation to the Agent List

Select the "**Enter url**" input field and enter the agent's URL and then press the "**enter**" button. The manager will then add the agent and will attempt to make a connection.

Note: If agent is installed on the "Execution Station", don't use local host address (i.e 127.0.0.1) to connect to agent, instead use "Execution Station" LAN IP.

- **Remove –** In order to remove an agent from the managed agents list select one or more agents and press on the "**Remove**" button. Once an agent is removed from the list, the user can not assign it to a test/scenario, and events dispatched from the agent will not be shown by the JRunner.
- **Synchronize** – In order to synchronize the agent with the local project, select the agents to be synchronized, and press on the "**Synchronize**" button.

  After pressing the "**Synchronize**" button the following occurs.

  1. The currently active automation project in the execution station is zipped.
  2. The zip is then transferred, using FTP protocol to the selected agents.
  3. Agents are then signaled to extract the project.
  4. The currently active SUT file and scenarios are set in the agents.
  5. During the process a progress panel is displayed to the user.
- **Refresh** - When pressed, the JRunner tries to reconnect to selected agents and fetch the agent information.
- **Close** -Closes the management dialog.

1.5.5 Assigning an Agent to a Scenario and/or Test

In order to associate a test and or scenario with an agent perform the following steps.

1. Select the test/scenario in the scenario tree.
2. Go to the "**Test Info**" tab.
3. In the "**Test Info**" tab, select the "**JSystem-Agents**" sub-tab.

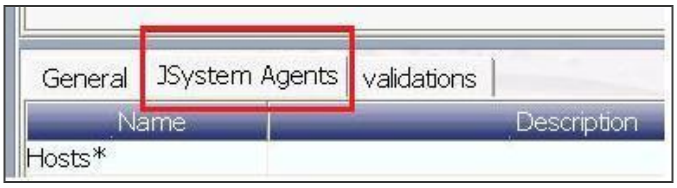


Figure 9: Assigning an Agent to the Host

4. Select the "**Value**" input field and the "**file selection**" button will appear.

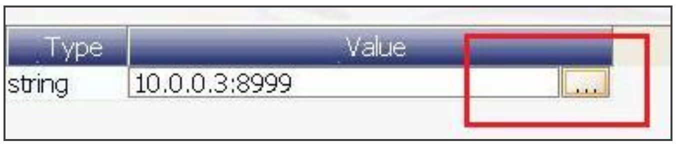


Figure 10: Setting the Value of the Host

5. After pressing on the "**Select File**" button, a selection dialog opens, now select the hosts that the scenario/test will run on and press the "**Save**" button.

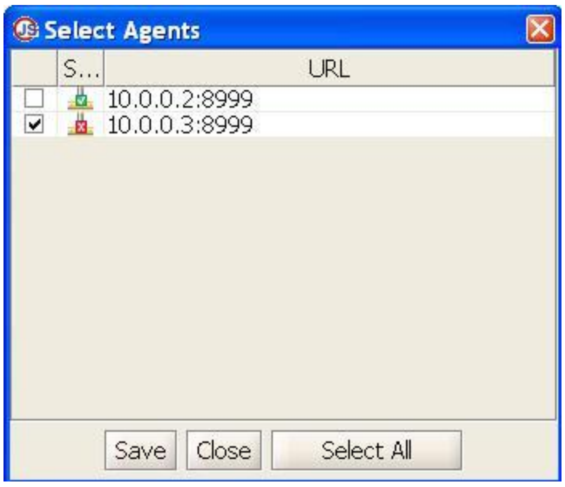
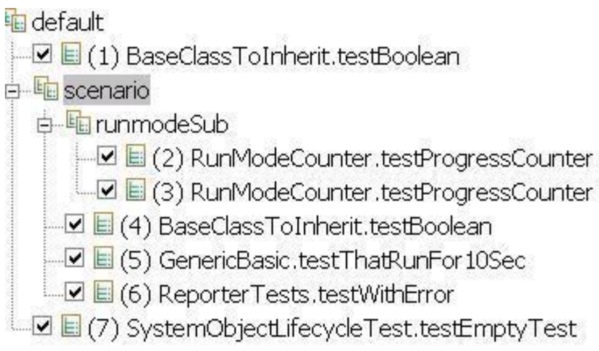


Figure 11: Select Agent Window

1.5.5.1 Assignation Notes

1. The top level assignments overrides assignments made to the offspring, as it is illustrated in the following scenario.



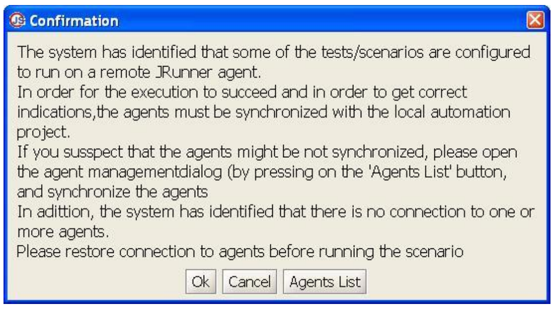
```
default
├ (1) BaseClassToInherit.testBoolean
├ scenario
│ ├ runmodeSub
│ │ ├ (2) RunModeCounter.testProgressCounter
│ │ └ (3) RunModeCounter.testProgressCounter
│ ├ (4) BaseClassToInherit.testBoolean
│ ├ (5) GenericBasic.testThatRunFor10Sec
│ └ (6) ReporterTests.testWithError
└ (7) SystemObjectLifecycleTest.testEmptyTest
```

Figure 12: Assigning agents to a Scenario

2. Assignments made to the “**scenarioPipi**” and override assignments made to the "**runmodeSub**" scenario.
3. The scenario "**runmodeSub**" is executed on the agents that are associated with the "**scenarioPipi**" scenario.
4. The root scenario cannot be assigned to run on a remote agent.

1.5.6 Running a Distributed Scenario

1. In order to run the scenario, press the "**Play**" button.
2. The system identifies that some of the tests/scenarios are configured to run are on a remote agent and shows the following message.



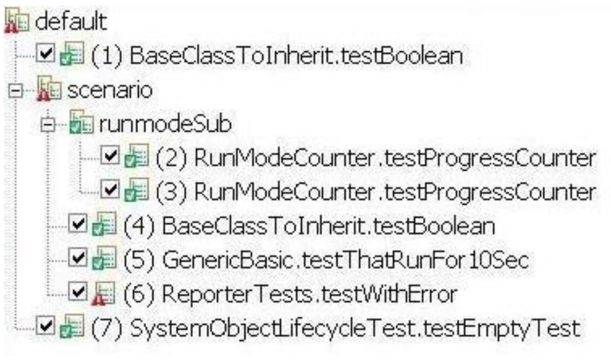
The system has identified that some of the tests/scenarios are configured to run on a remote JRunner agent.
In order for the execution to succeed and in order to get correct indications,the agents must be synchronized with the local automation project.
If you susspect that the agents might be not synchronized, please open the agent managementdialog (by pressing on the 'Agents List' button, and synchronize the agents
In adittion, the system has identified that there is no connection to one or more agents.
Please restore connection to agents before running the scenario

Figure 13: Confirmation Message

3. The user can open the "**Agents Management**" dialog by pressing on "**Agents List**" button.
4. If the system identifies that the scenario was configured to run on an agent that the JRunner cannot connect to, a message is sent to the user.

Once the scenario starts to run, the user can view the progress in two places:

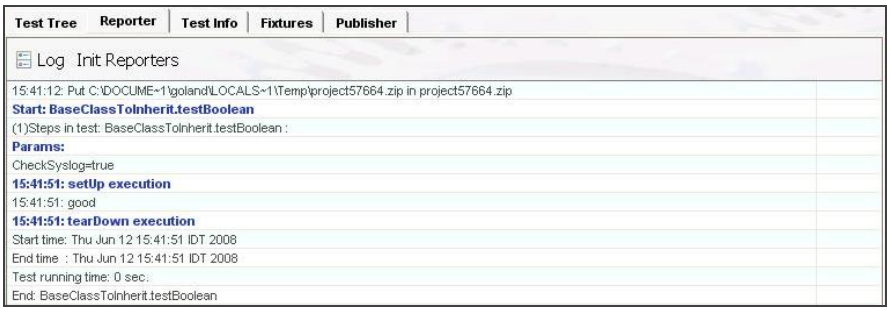
1. Scenario tree



```
default
  (1) BaseClassToInherit.testBoolean
  scenario
    runmodeSub
      (2) RunModeCounter.testProgressCounter
      (3) RunModeCounter.testProgressCounter
    (4) BaseClassToInherit.testBoolean
    (5) GenericBasic.testThatRunFor10Sec
    (6) ReporterTests.testWithError
  (7) SystemObjectLifecycleTest.testEmptyTest
```

Figure 14: Scenario Tree

2. **Reporter Tab** - A reporter table is created for each of the agents during the run, the icon on the tab indicates the execution state and the table shows the log events received from the agents.

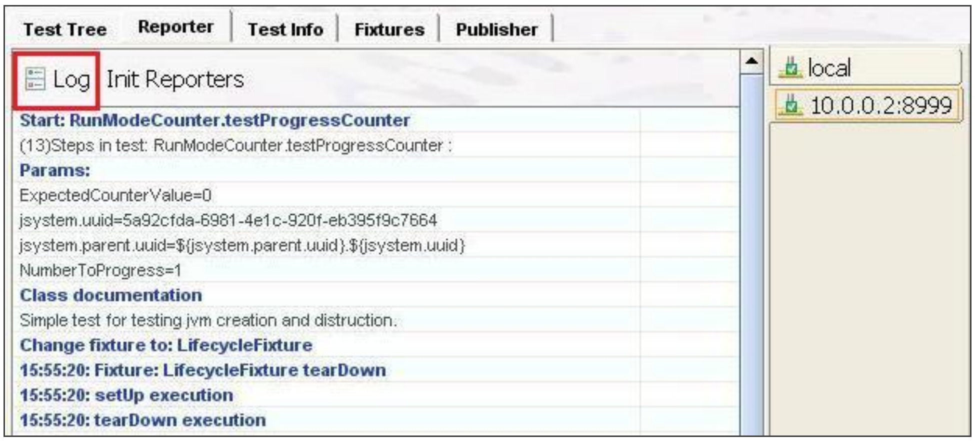


```
15:41:12: Put C:\DOCUME~1\goland\LOCALS~1\Temp\project57664.zip in project57664.zip
Start: BaseClassToInherit.testBoolean
(1)Steps in test: BaseClassToInherit.testBoolean :
Params:
CheckSyslog=true
15:41:51: setUp execution
15:41:51: good
15:41:51: tearDown execution
Start time: Thu Jun 12 15:41:51 IDT 2008
End time  : Thu Jun 12 15:41:51 IDT 2008
Test running time: 0 sec.
End: BaseClassToInherit.testBoolean
```

Figure 15: Reporter Tab

1.5.7 Analyzing Results

Once the test execution has completed, the user can now press on the "**Log**" button in the agents "**Reporter**" tab and then analyze the execution results.

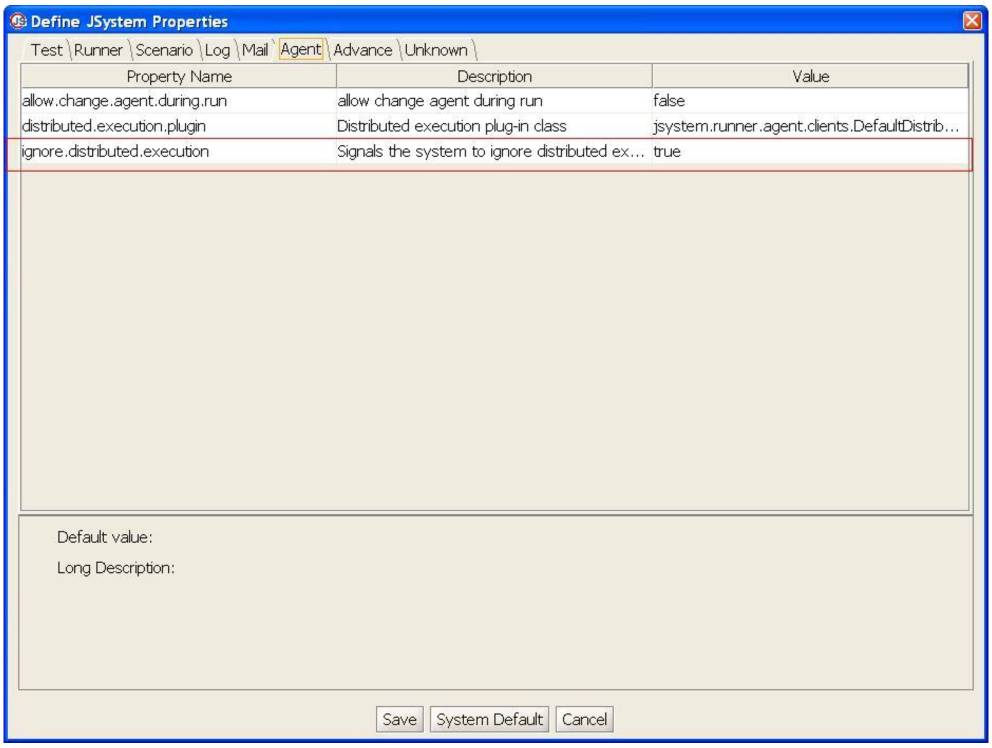


```
Start: RunModeCounter.testProgressCounter
(13)Steps in test: RunModeCounter.testProgressCounter :
Params:
ExpectedCounterValue=0
jsystem.uuid=5a92cfda-6981-4e1c-920f-eb395f9c7664
jsystem.parent.uuid=${jsystem.parent.uuid}.${jsystem.uuid}
NumberToProgress=1
Class documentation
Simple test for testing jvm creation and distruction.
Change fixture to: LifecycleFixture
15:55:20: Fixture: LifecycleFixture tearDown
15:55:20: setUp execution
15:55:20: tearDown execution
```

Figure 16: JReporter Tab

1.5.8 Disabling a Distributed Execution

In order to disable the distributed execution, select Tools>Define JSystem Properties, click on the "**Agent**" tab, and set the "**ignore.distributed.execution**" to true.



Property Name	Description	Value
allow.change.agent.during.run	allow change agent during run	false
distributed.execution.plugin	Distributed execution plug-in class	jsystem.runner.agent.clients.DefaultDistrib...
ignore.distributed.execution	Signals the system to ignore distributed ex...	true

Default value:

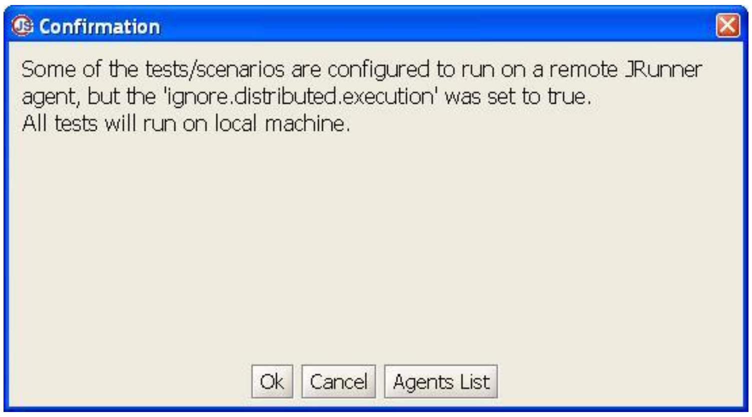
Long Description:

Figure 17: JSystem Properties Window

1.5.8.1 Confirmation Dialog

The “**ignore.distributed.execution**” property has been added in order to enable easy planning and developing of the scenario. While working on the scenario, the user can set the property value to true, and all tests will run locally.

When the property value is set to true, and the scenario is run, the following message is appears.`



Confirmation

Some of the tests/scenarios are configured to run on a remote JRunner agent, but the 'ignore.distributed.execution' was set to true.
All tests will run on local machine.

Figure 18: Confirmation Dialog

Once the scenario is ready, the user can set the property to false and the scenario will be executed in a distributed manner.

1.6 JUnit 4 Support for Test Authoring

JUnit 4, was originally released two years ago, it brought about a new cleaner style of writing test cases. Tests written in the JUnit 4 style no longer depend on a specific method naming convention (starting each method name with the word *test*, calling the set up and tear down methods *setUp* and *tearDown*). Instead, the user can use annotations to mark methods as tests as well as additional methods to be run before or after test methods.

This latest version of JSystem adds s JUnit 4 support. JSystem still supports the Junit 3 test style.

```
public class MyTestCase extends SystemTestCase {
    public MyTestCase() {
        super();
        setParentFixture(BasicFixture.class);
    }
    public void setUp() {
        report.step("MyTestCase setUp");
    }
    public void tearDown() {
        report.step("MyTestCase tearDown");
    }
        public void testSomething() {
        report.step("Testing something");
    }
}
```

Table 8: JSystem test based on JUnit 3 Code Example

The user can use the cleaner JUnit 4 style in the following manner.

```
public class MyTestCase extends SystemTestCase4 {

    public MyTestCase() {

        super();

        setParentFixture(BasicFixture.class);

    }

      @Before

    public void before() {

        report.step("MyTestCase setUp");

    }

        @After

    public void after() {

        report.step("MyTestCase tearDown");

    }

    @Test

    public void something() {

        report.step("Testing something");

    }

}
```

Table 9: JUnit MyTestCase Code Example

The changes that were made to the code in transition from JUnit 3 to JUnit 4 style are detailed as follows.

- The test case class extends the ***SystemTestCase4,*** instead of the ***SystemTestCase***.
- The ***@Before* and *@After*** annotations mark methods that are run during the test set up and tear down respectively. All the ***@Before*** methods are called before each test is run and all the ***@After*** methods are called when it ends.
- Test method names do not need to start with the word ***test***. The test author can name the test as required, as long as the test is annotated with the ***@Test*** annotation.
- Other JUnit 4 annotations:
    a. **@BeforeClass** – will be executed before the @Before annotation
    b. **@AfterClass** - will be executed after the @After annotation
    c. **@Igone** – test will not be executed

All the old test written in the JUnit 3 style test cases run without any problems. Only tests deriving from "***SystemTestCase4"*** will be treated as JUnit 4 style tests.

1.7 Jython Test Support

Jython (www.jython.org) is a Java implementation of the Python scripting language (www.python.org). Its concise syntax and dynamic nature makes it a perfect fit for writing test cases and the latest version of JSystem allows the test author to use it to write JSystem system tests.

A test in Jython is written as a class in a ***.py** file using Jython's ***unittest***, a library for running unit tests in Jython and Python, similar to JUnit for Java.

The following example test is written in Jython using an imaginary system object that can check for the existence of a file in a directory:

```
import unittest
from jyutils import *
class FileTests(SystemTestCase):
    __parameters__ = dict(
        directory = Parameter(),
        file = Parameter()
    )
        def setUp(self):
        self.sysobj = system.getSystemObject("fileChecker")
        def test1(self):
        report.report("Directory : %s" % self.directory)
        report.report("File : %s" % self.file)
         self.assertTrue(self.sysobj.isFileInDir(self.file,
self.directory))
```

Table 10: Jython Code Example

1.7.1 Writing Jython Tests

Each Jython test file should start with the *from* ***jyutils import **** line in order to allow the tests access to various JSystem features. Each file should contain one or more class, each extending **"SystemTestCase"**. In a fashion similar to JUnit (version 3) the tests are described in methods whose names start with the word ***test***, with ***setUp*** called before each test method is run and ***teardown*** called after each test.

You can define parameters for the test that will be displayed in the JRunner user interface by creating a __***parameters***__ static data member, which is a ***dict*** of parameter names mapped to *Parameter* objects. The parameter values are accessible as data members in the test methods themselves (see references to ***self.file*** and ***self.directory*** above).

Jython test files should be put in a subdirectory of your current project path alongside the *.class files of your Java tests.

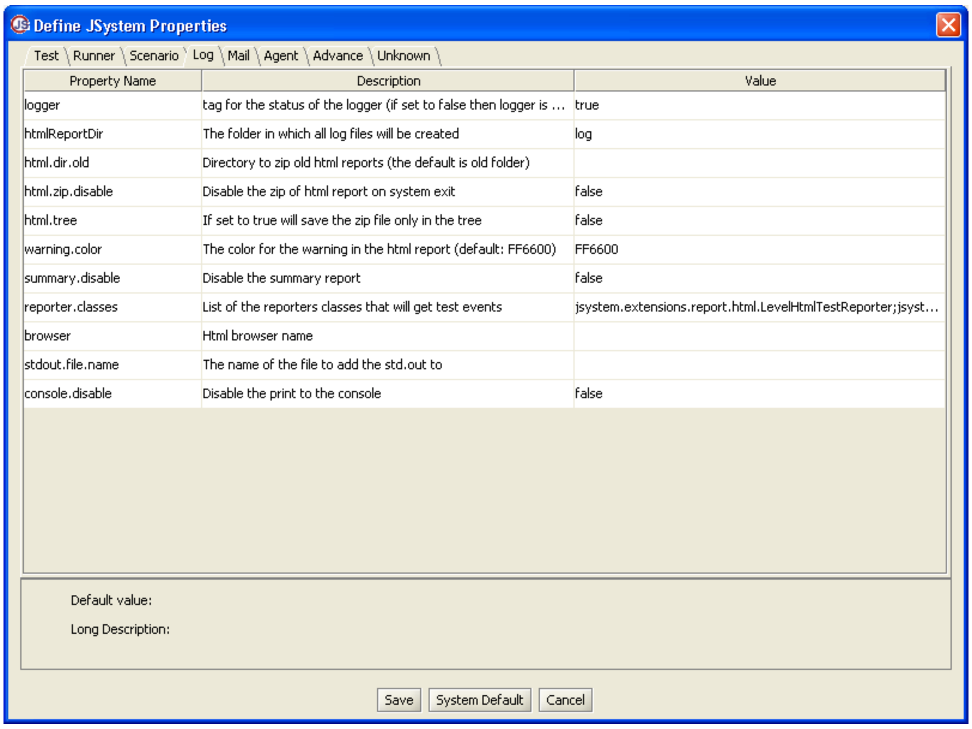
You can create scenarios that mix Jython test cases and regular JUnit test cases. In order to use the new Jython capabilities, you need to add the following line to you ***jsystem.properties*** file:

```
script.engines=jsystem.framework.scripts.jython.JythonScriptEngin
e
```

Table 11: JSystem Properties File

1.8 JSystem Properties Dialog

The JSystem "**Properties Dialog**", enable the user to configure the JSystem properties file.



Property Name	Description	Value
logger	tag for the status of the logger (if set to false then logger is ...	true
htmlReportDir	The folder in which all log files will be created	log
html.dir.old	Directory to zip old html reports (the default is old folder)	
html.zip.disable	Disable the zip of html report on system exit	false
html.tree	If set to true will save the zip file only in the tree	false
warning.color	The color for the warning in the html report (default: FF6600)	FF6600
summary.disable	Disable the summary report	false
reporter.classes	List of the reporters classes that will get test events	jsystem.extensions.report.html.LevelHtmlTestReporter;jsyst...
browser	Html browser name	
stdout.file.name	The name of the file to add the std.out to	
console.disable	Disable the print to the console	false

Default value:

Long Description:

Figure 19: JSystem Properties Dialog

1.8.1 Using the JSystem Properties Dialog

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

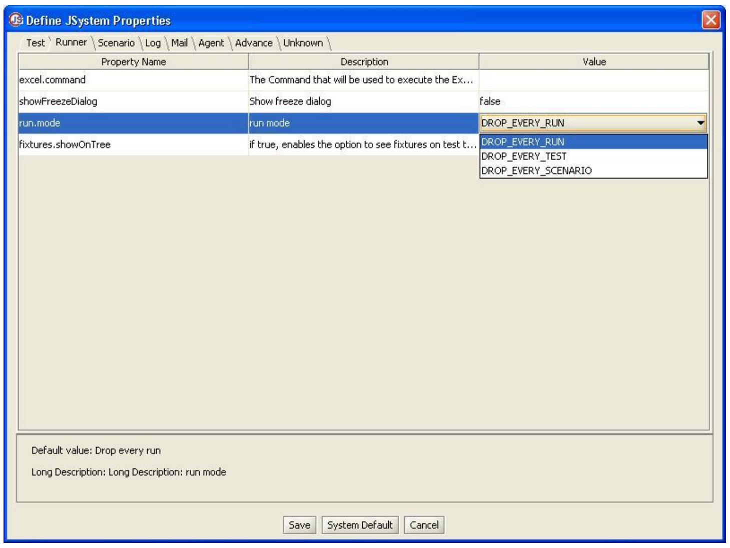
1.8.1.1 JSystem Properties Table

- **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.



Property Name	Description	Value
excel.command	The Command that will be used to execute the Ex...	
showFreezeDialog	Show freeze dialog	false
run.mode	run mode	DROP_EVERY_RUN
fixtures.showOnTree	if true, enables the option to see fixtures on test t...	

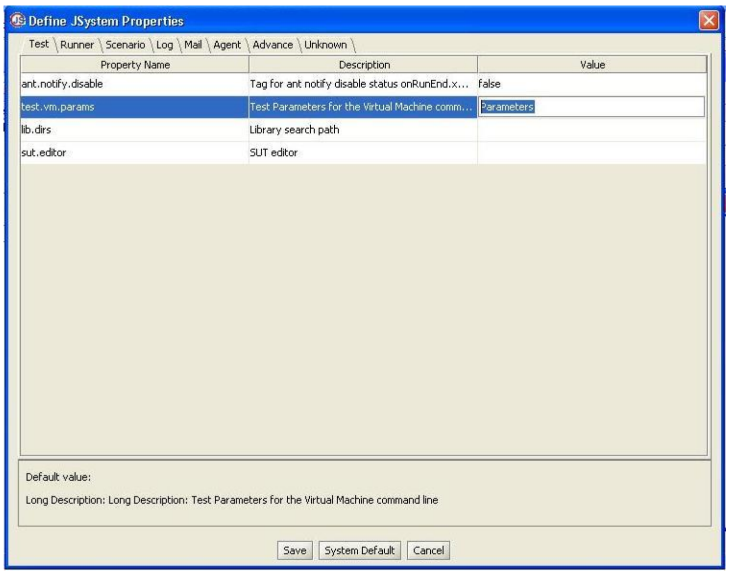
DROP_EVERY_RUN
DROP_EVERY_TEST
DROP_EVERY_SCENARIO

Default value: Drop every run

Long Description: Long Description: run mode

Figure 20: Properties Dialog Runner Tab

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



Property Name	Description	Value
ant.notify.disable	Tag for ant notify disable status onRunEnd.x...	false
test.vm.params	Test Parameters for the Virtual Machine comm...	Parameters
lib.dirs	Library search path	
sut.editor	SUT editor	

Default value:

Long Description: Long Description: Test Parameters for the Virtual Machine command line

Figure 21: Properties Dialog – Setting the Parameters

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

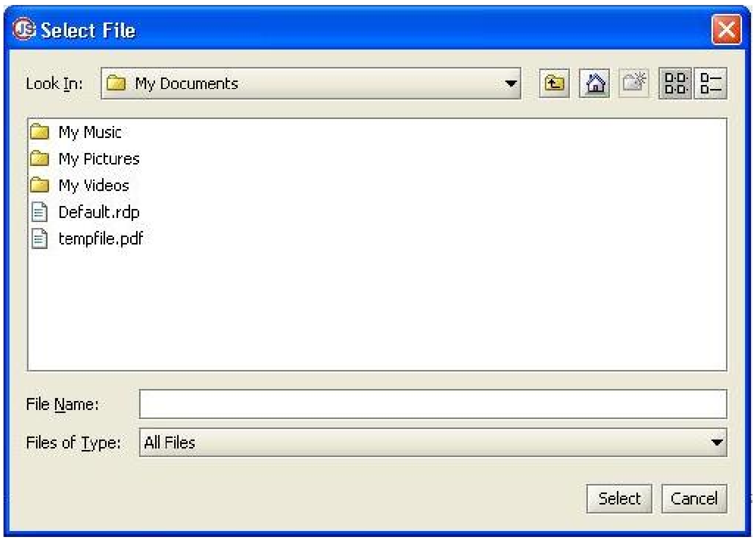


Figure 22: Properties Dialog - Selecting a File

- **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, when the property is used, it's purpose, including examples if needed and a detailed explanations of the value options.

1.8.2 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 "**jsystem.properties"** "(Under Runner installation folder).

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

When the user pressed 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.

Usually, all of the properties are saved into "**jsystem.properties"** file under the installation JRunner directory.

In the event that the user chooses to restore the defaults, the "**jsystem.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 "**jsystem.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.