# Version 5.6

## Release Notes

## Disclaimer

## Table of Contents

# 1. Purpose of this Document

These release notes describe the new features and bug fixes for version 5.6 of ***JSystem Automation Platform***.

This document along with previous releases documentation and JSystem User and Developer Guide holds a complete documentation of JSystem Automation Platform.

As of this release, previous release documentation is attached as part of the release documentation so the release comes with complete documentation.

## 1.1 New Features Summary

JSystem 5.6 includes features that span over all of the automation development lifecycle and users functionality. The new features include enhancements for the following components:

- Scenario studio
- Parameters panel
- Flow control
- Html reporter
- Sut
- Assets tree
- Multiple Scenarios Execution

## 1.2 JSystem 5.6 Licensing

JSystem 5.6 is released under Apache version 2.0 license.

JSystem 5.6 uses many open source projects. A complete list of all projects and their licenses can be found in JSystem Trac.

Note that JSystem core uses only projects with BSD like licenses (BSD, Apache 2.0, LGPL etc.) with the exception of java2html (which uses GPL).

JSystem uses java2html to display the test code in the reporter and this function is optional. JSystem 5.6 installation comes without java2html. If you wish to use it simply download it from http://www.java2html.com, place it under your thirdparty/lib and add it to your classpath.

# 2. Scenario Studio Enhancements

Enhancements were done to the scenario studio section to allow quicker and easier operations on scenario editing and allowing more customization to the scenario elements.

## 2.1 Copy Paste

One of the most important enhancements in this version is the Copy-Paste-Cut feature.

This allows re-using of already created components by duplicating and moving them around in the scenario or between scenarios.

The user can now mark any component s/he wants to duplicate/move/cut and copy or cut it.

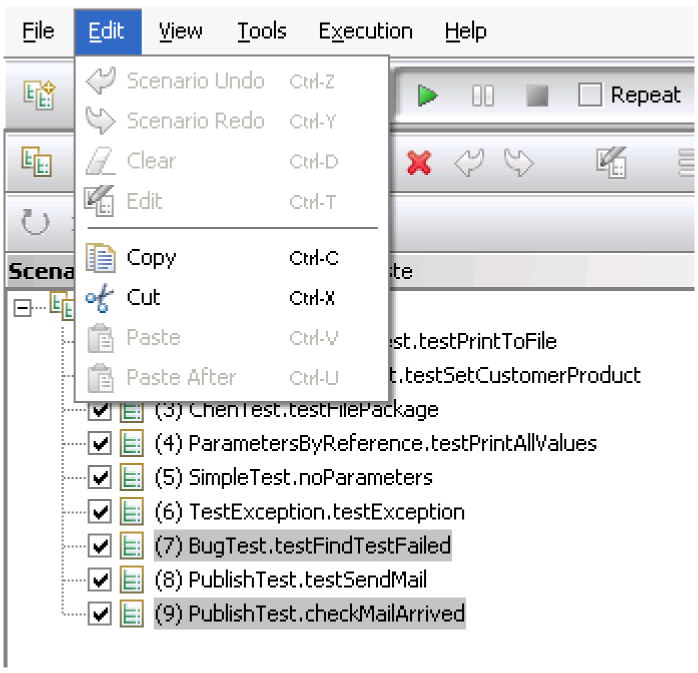
The component is saved in the memory and can be pasted later on in any open scenario; as long as the JRunner was not closed (closing the JRunner clears the clipboard).

The Copy\Cut operation only marks the components to copy, with no regards to containers structure or tests parameters values.
The actual components structure and parameters values will be determined during the Paste operation. Therefore, any changes made to the container\tests after selecting Copy\Cut will be included in the Paste operation.

Example:

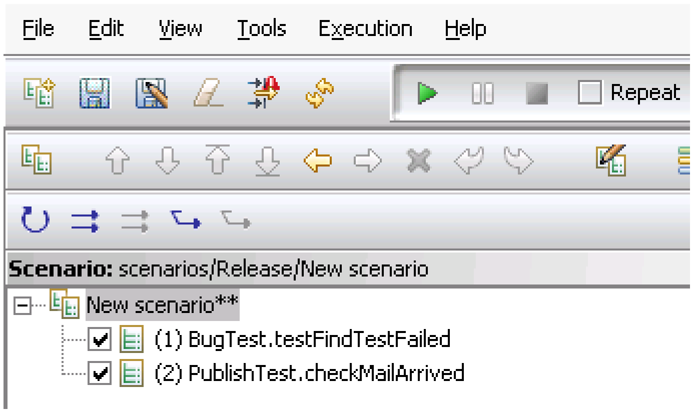
In the following screen capture, two tests were marked.

After that the user can either select the Copy/Cut from the Edit menu or use the standard shortcuts (Ctrl+c/Ctrl+x).


File
Edit
View
Tools
Execution
Help
Scenario Undo Ctrl-Z
Scenario Redo Ctrl-Y
Clear Ctrl-D
Edit Ctrl-T
Copy Ctrl-C
Cut Ctrl-X
Paste Ctrl-V
Paste After Ctrl-U
Repeat
Scena
te
st.testPrintToFile
t.testSetCustomerProduct
(3) ChenTest.testFilePackage
(4) ParametersByReference.testPrintAllValues
(5) SimpleTest.noParameters
(6) TestException.testException
(7) BugTest.testFindTestFailed
(8) PublishTest.testSendMail
(9) PublishTest.checkMailArrived

After that, the tests are saved to the memory and can be pasted anywhere, even in a new scenario.

In the following screen capture a new scenario was created and the paste was chosen (again, from the Edit menu or by using Ctrl+v).

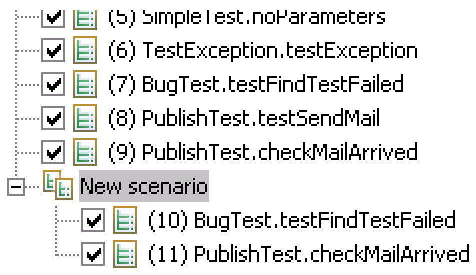

File
Edit
View
Tools
Execution
Help
Repeat
Scenario: scenarios/Release/New scenario
New scenario**
(1) BugTest.testFindTestFailed
(2) PublishTest.checkMailArrived

Paste After:

In addition to the "Paste" option, there is an option to "Paste After" (Ctrl+u).

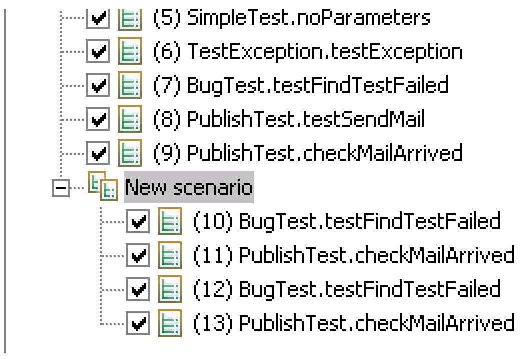
This option is only relevant for container targets, i.e. scenarios and flows elements, allowing the user to select whether s/he wants to paste the content **inside** the container (using the "Paste") or **after** the component (using the "Paste After").

In the following screen shot, the "New Scenario" target is selected:



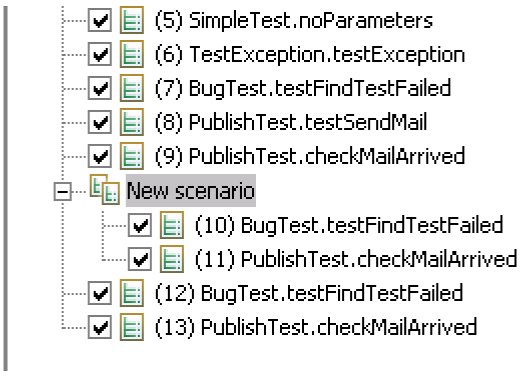
```
(5) SimpleTest.noParameters
(6) TestException.testException
(7) BugTest.testFindTestFailed
(8) PublishTest.testSendMail
(9) PublishTest.checkMailArrived
New scenario
    (10) BugTest.testFindTestFailed
    (11) PublishTest.checkMailArrived
```

Using "Paste" will place the copied content inside the scenario, as follows:



```
(5) SimpleTest.noParameters
(6) TestException.testException
(7) BugTest.testFindTestFailed
(8) PublishTest.testSendMail
(9) PublishTest.checkMailArrived
New scenario
    (10) BugTest.testFindTestFailed
    (11) PublishTest.checkMailArrived
    (12) BugTest.testFindTestFailed
    (13) PublishTest.checkMailArrived
```

While using "Paste After" will place the copied content after the scenario, as follows:



```
(5) SimpleTest.noParameters
(6) TestException.testException
(7) BugTest.testFindTestFailed
(8) PublishTest.testSendMail
(9) PublishTest.checkMailArrived
New scenario
    (10) BugTest.testFindTestFailed
    (11) PublishTest.checkMailArrived
(12) BugTest.testFindTestFailed
(13) PublishTest.checkMailArrived
```

Special Cases:

- In case of an illegal Pasting action which is legal to be pasted after the selected object (for example, pasting a SwitchCase into a SwitchCase), the Clipboard object/tests will be added AFTER the selected object and not in it.

## 2.2 Scenario Name Hook

Scenarios names are unique. In a single project, a duplicate scenario name is not allowed.

This imposes some limitations on group working. One case is when the same project is used by several end users who want to use the same scenario name. Other case is when two projects are merged.

In other cases, the user might want to manipulate the scenario native name and add more data to it.

Thus, we have added the ability to manipulate the scenario name (on creation) in order to have a more customizable scenario names.

In addition, JSystem 5.6 supports new attribute for Scenario – Project Name. Using this attribute users can mark scenarios is being part of the same project. Note that this is marking only. It has no operational functionality.

The scenario name manipulation is done by creating a class of your own, extending the *ScenarioNameHook* interface and implementing its two methods: *getProjectName* and *getScenarioId*.

The project name is a string related to the scenario, which can be viewed using the Tools->Project name described later on.

The scenario ID is a string related to the scenario which will be suffixed to the scenario name.

Example:

```
package jsystem.extensions.scenarionamehook;

public class ExampleHook implements ScenarioNameHook {
	static int index = 0;
	@Override
	public String getProjectName() {
		return "Default Project";
	}

	@Override
	public String getScenarioId() {
		index +=1;
		return "_"+index;
	}
}
```

After creating a *ScenarioNameHook*, the implementation class should be specified in the JSystem.properties. Using the JSystem properties dialog, under the "Advanced" tab, look for the "scenario.name.hook" key and set as following:

scenario.name.hook=jsystem.extensions.scenarionamehook.ExampleHook

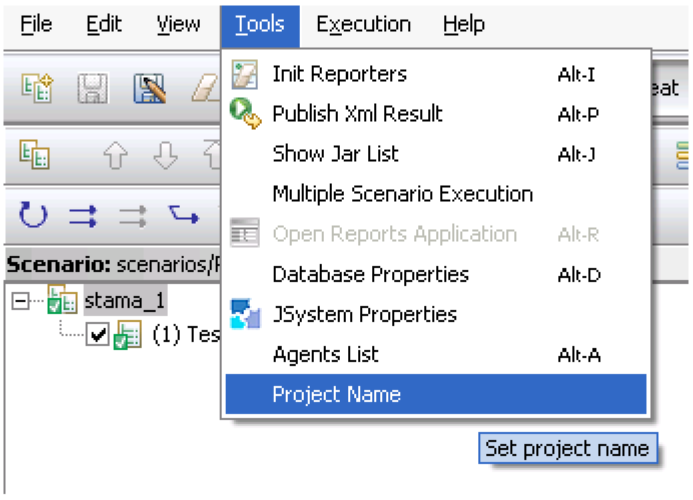
The resulting scenario name will be constructed as follows:

<User scenario name> + <id>

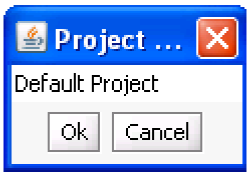
Therefore, for the above example, if the user created a scenario names "test", the result will be a scenario called "test_1".

The next scenario created will be "<scenario name>_2" and so on.

The project name is not visible and can be viewed or changed using the "Tools" menu, as shown in the following screen capture:



After selecting this option, the following dialog will appear, allowing the user to edit the related project name for the current scenario.



**Note: the menu option will appear only if the user has specified a valid ScenarioNameHook in the jsystem.properties file.**

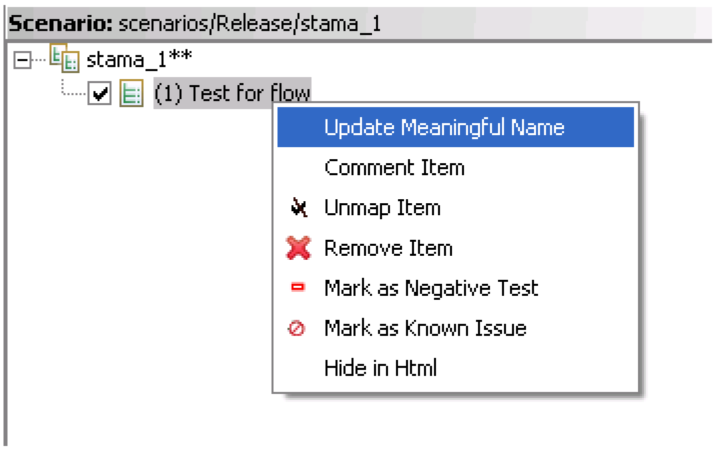
## 2.3 Meaningful Name for Tests

In previous JSystem version we have added the ability to set a meaningful name for each test (from code) in order to have more human readable names in the JRunner.

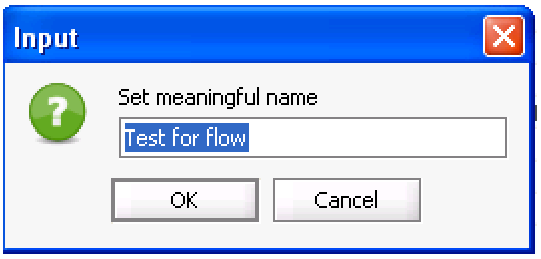
In some cases, this is not enough and there is a need to also set a different name for the same test, inside a scenario.

Therefore, we have added an option to set a meaningful name for a test from the Scenario Editor.

The option is visible in the right click menu as seen below:



After selecting the "Update Meaningful Name" menu item, a popup will appear:



Enter the meaningful name for this test and press "OK".

"Cancel" will cancel the operation.

**Note: in order to remove a meaningful name from a test, delete the content of the Input dialog and select "OK".**

# 3. Parameters Panel

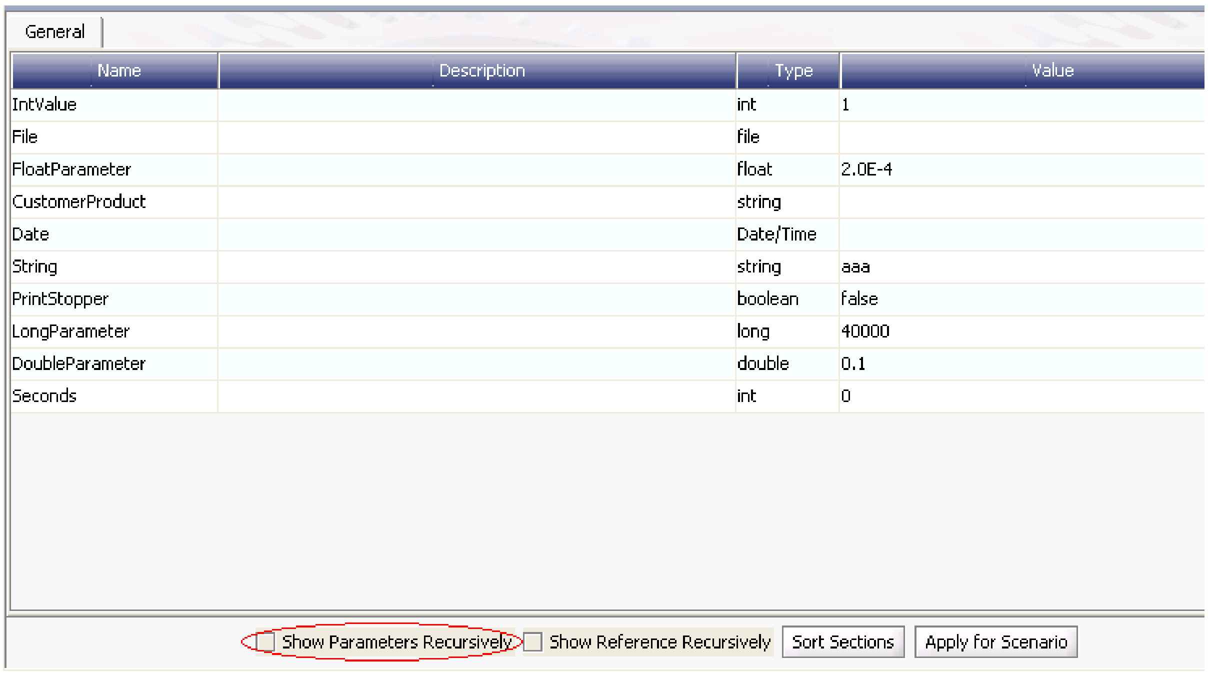
A few changes were done to the parameters panel in order to allow recursive parameters value applying and more customization to the ordering.

## 3.1 Apply Scenario Recursively

Until JSystem 5.6, parameters values could be set for a specific test or applied for the entire scenario (using the "Apply for Scenario" button).

However, there was no option to apply the value recursively to all sub scenarios as well.

As of JSystem 5.6, we added new checkbox to the parameters panel, "Show Parameters Recursively", as shown below:

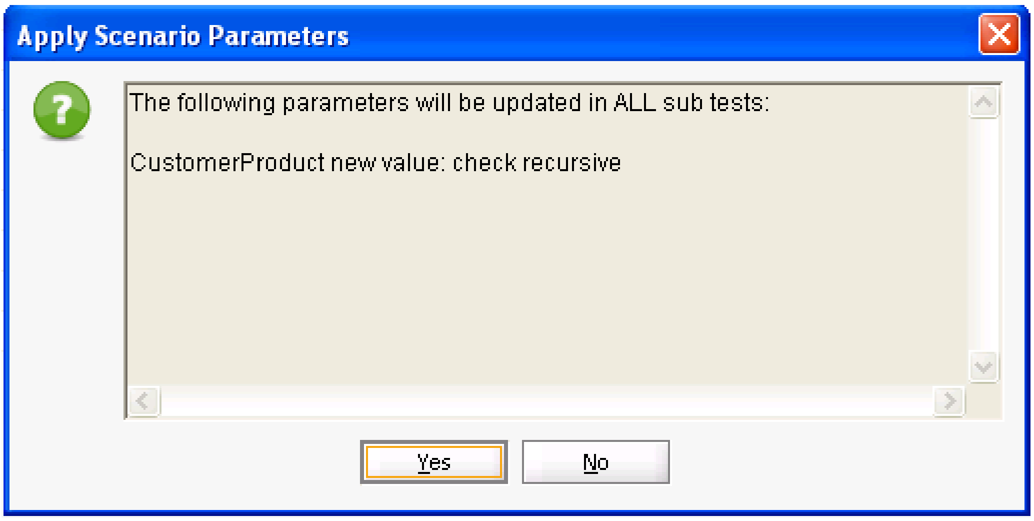


| Name | Description | Type | Value |
|---|---|---|---|
| IntValue | | int | 1 |
| File | | file | |
| FloatParameter | | float | 2.0E-4 |
| CustomerProduct | | string | |
| Date | | Date/Time | |
| String | | string | aaa |
| PrintStopper | | boolean | false |
| LongParameter | | long | 40000 |
| DoubleParameter | | double | 0.1 |
| Seconds | | int | 0 |

Checking this checkbox will display all visible parameters in all tests in the selected scenario and sub-scenarios (when selecting a test, rather than a scenario, the checkbox will be grayed out).

While this checkbox is selected, the "Apply for Scenario" button will apply the changed value to all sub-scenarios as well.

The following message will be displayed:



The following parameters will be updated in ALL sub tests:

CustomerProduct new value: check recursive

(Note the "ALL" emphasize).

## 3.2 Return Parameters

Up until JSystem 5.6, "return parameters" were also needed to be defined as "include parameters"; causing them to be visible in the Parameters panel, although in many cases they were only needed as internal parameters.

As of JSystem 5.6 this limitation was removed so return parameters no longer have to be visible in the parameters panel.

## 3.3 Parameters Order Default Setting

The parameters panel can be sorted in any column.

The order can be:

a) Alphabetical/Numerical ascending.

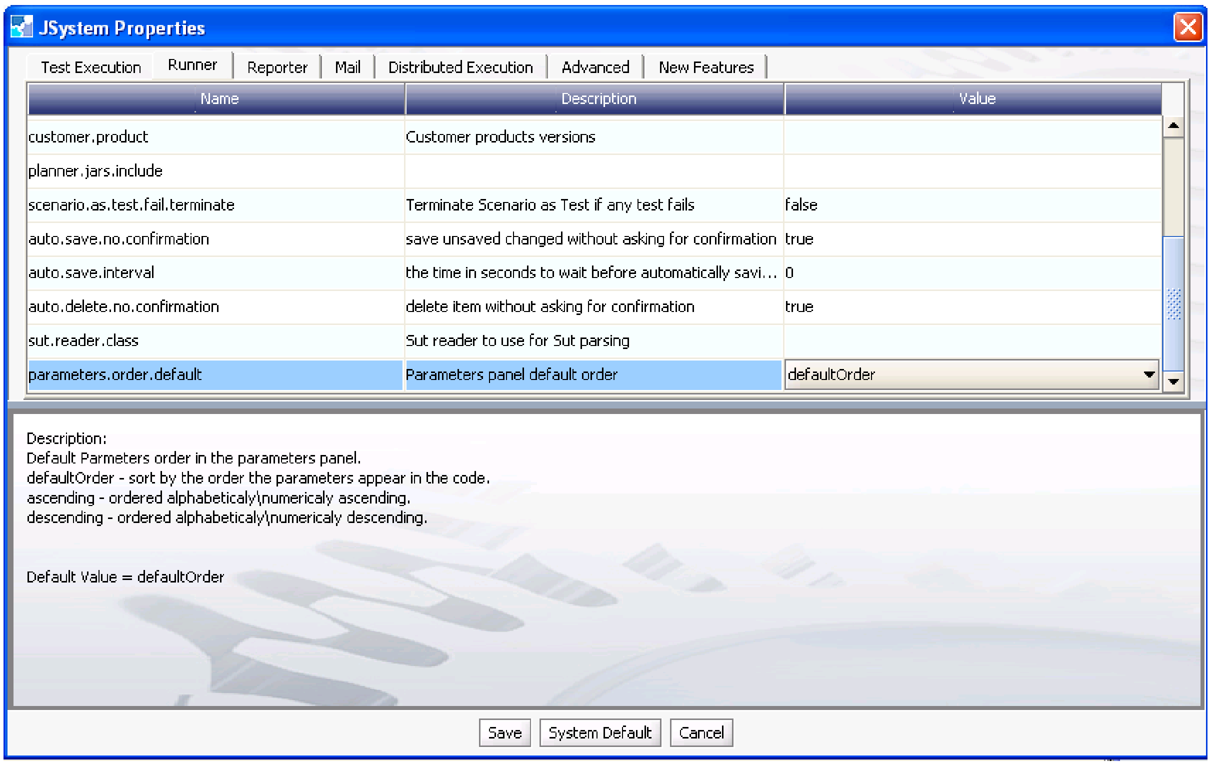
b) Alphabetical/Numerical descending.

c) Natural order as appears in the code.

The default order was fixed to Natural order.

In order to allow more customization, the user can now alter the default order of parameters.

This is done using the JSystem properties dialog, under the "Runner" tab; look for the "parameters.order.default" key:



| Name | Description | Value |
|---|---|---|
| customer.product | Customer products versions | |
| planner.jars.include | | |
| scenario.as.test.fail.terminate | Terminate Scenario as Test if any test fails | false |
| auto.save.no.confirmation | save unsaved changed without asking for confirmation | true |
| auto.save.interval | the time in seconds to wait before automatically savi... | 0 |
| auto.delete.no.confirmation | delete item without asking for confirmation | true |
| sut.reader.class | Sut reader to use for Sut parsing | |
| parameters.order.default | Parameters panel default order | defaultOrder |

Description:
Default Parmeters order in the parameters panel.
defaultOrder - sort by the order the parameters appear in the code.
ascending - ordered alphabeticaly\numericaly ascending.
descending - ordered alphabeticaly\numericaly descending.

Default Value = defaultOrder

# 4. Flow Control Enhancements

## 4.1 User Documentation for Flow Elements

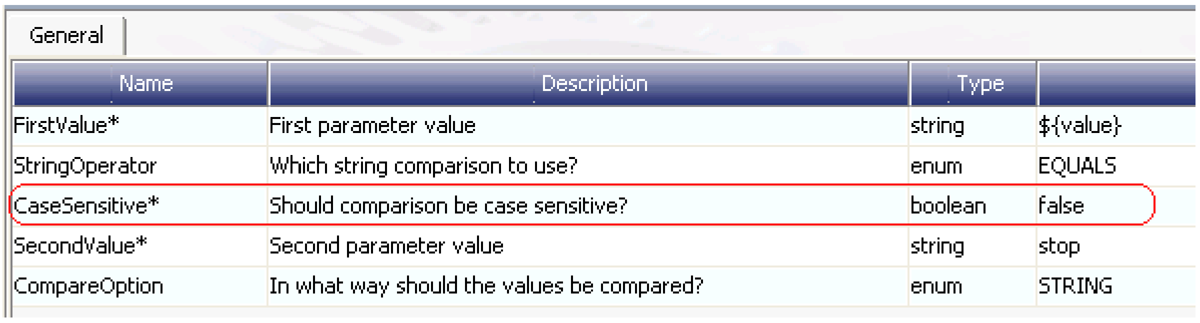
Till JSystem 5.6, User Documentation was only available for tests and scenarios, allowing the user to add notes for each component.

This ability is now also available for flow elements in the same manner.

## 4.2 Case Sensitive "if"

In JSystem 5.5, the "If" component was enhanced, allowing support for both a Textual and a Numerical comparison.

In JSystem 5.6 the Textual comparison can also be case sensitive, as appears in the below example:



General

| Name | Description | Type | |
|---|---|---|---|
| FirstValue* | First parameter value | string | ${value} |
| StringOperator | Which string comparison to use? | enum | EQUALS |
| CaseSensitive* | Should comparison be case sensitive? | boolean | false |
| SecondValue* | Second parameter value | string | stop |
| CompareOption | In what way should the values be compared? | enum | STRING |

## 5. Html Reporter Enhancements

The Html reporter main purpose is to be a fancy log of scenario execution. Nevertheless, the Html report is being also viewed by people without technical orientation. In JSystem 5.6 we have added some capabilities which will help the developers of the automation project make the Html report more "managers friendly".

### 5.1 Hide in Html Full Functionality

In prior JSystem versions, the "Hide in Html" flag was added, allowing the user to select tests that should not appear in the Html report, as long as they pass.

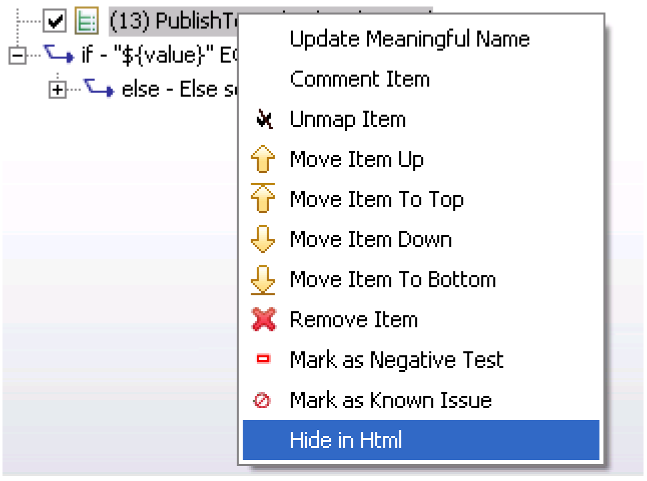
In JSystem 5.6 this ability was extended so that tests marked as Hide in Html that passes:

1) Will not be displayed in the Html report.
2) Will not be counted in statistics.
3) Tests related files will be deleted from the file system.

Tests that do not pass will act normally, as if they were not marked with this flag.

**Note: since tests that pass are not saved in the file system, and mostly these tests are of less interest since they passed, this option can be used for optimizing file system usage.**

To use the Hide in Html flag, mark the requested test(s) and select the "Hide in Html" from the right click menu, as shown below:



## 5.2 CSS Tags for Html Elements

In continue to prior JSystem 5.6 Html reporter customization, more CSS tags were added to the html.

All tags are defined in the default.css file (located in the runner\log\current folder).

The default.css file can be replaced by a list of css files which will be incorporated in the Html report by going to the jsystem properties dialog, "Reporter" tab, under "css.path." key.

css paths should be separated with a semicolon, css files are searched in the classpath and are added to the html in their order in the css.path property.

New CSS tags:

1) Bread crumbs (Test bread crumbs inside test information table).
2) Test information table (the table that appears on the upper right side of a test report).
3) Test parameters in the Test information table.
4) Test class documentation in the Test information table.
5) Test documentation in the Test information table.
6) Test user documentation in the Test information table.
7) Time stamps on report lines.

# 6. Sut Enhancements

The Sut file is a XML file representing the system under test.

In complex systems or test setups, with a lot of SystemObjects, the Sut file becomes complex and contains many tags.

In order to use smaller and more efficient tags a few enhancements exist.

One alternative is using reference in the Sut for existing SystemObjects, as explained in the JSystem Framework Services guide, under "Reference Support" section.

Another option is using your own Sut Reader mechanism as explained below.

## 6.1 Sut Reader Mechanism

The SutReader mechanism was added In order to allow a customized parsing for the Sut.

Starting JSystem 5.6 it is possible to add your own implementation for parsing the Sut XML file, allowing manipulations and modifications to different tags in the XML.

This allows placing your own "place holders" tags in the Sut which will be processed by your SutReader during runtime.

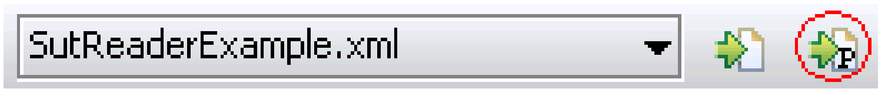
Usage

Create a SutReader implementation, by creating a new class implementing the jsystem.framework.sut.SutReader interface.

Add your SutReader class. Using the JSystem properties dialog, under the "Runner" tab, look for the "sut.reader.class" property.

After setting the "sut.reader.class" property, the JRunner needs to restart.

After that, a new button will be added to the JRunner.



(Note that this button is similar to the "Edit Sut" button, only contains a small "p" letter (indicating a **p**rocessed Sut).

This button allows viewing the Processed Sut file (after it is being processed by your SutReader) in order to make sure the processed Sut is processed correctly.

The processed Sut represents the Sut which will be used during test execution. No modifications are done to the original Sut file.

**Note: the button is named "View Processed Sut", since it only allows viewing the processed Sut file.**

**The created file is a temporary file and modifications done on this file are not saved.**

**Modifications should only be done on the original Sut file using the "Edit Sut" button.**

<u>Example</u>

(This example is available in: jsystem.framework.sut.ExampleReader).

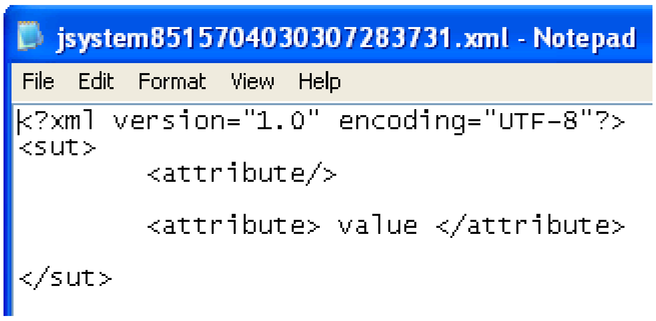
Let's say we want to add some attribute to an XML tag on test execution.

Bellow is an example for a simple SutReader implementation that search the Sut file for all tags named "attribute" and adds a "number" attribute to them with an incrementing number as its value.

```
public class ExampleReader implements SutReader {

	@Override
	public Document getDocument(File sutXml) throws Exception {
		Document doc =  XmlUtils.getDocumentBuilder().parse(sutXml);
		return manipulateDocument(doc);
	}

	@Override
	public Document getDocument(InputStream sutInputStream) throws
Exception {
		Document doc =
	XmlUtils.getDocumentBuilder().parse(sutInputStream);
		return manipulateDocument(doc);
	}

	private Document manipulateDocument(Document doc){
		ArrayList<Element> tags =
	XmlUtils.getElementsByTag("attribute", doc.getDocumentElement());
		int number = 0;
		for (Element tag : tags){
			tag.setAttribute("number", ++number + "");
		}

		return doc;
	}

}
```

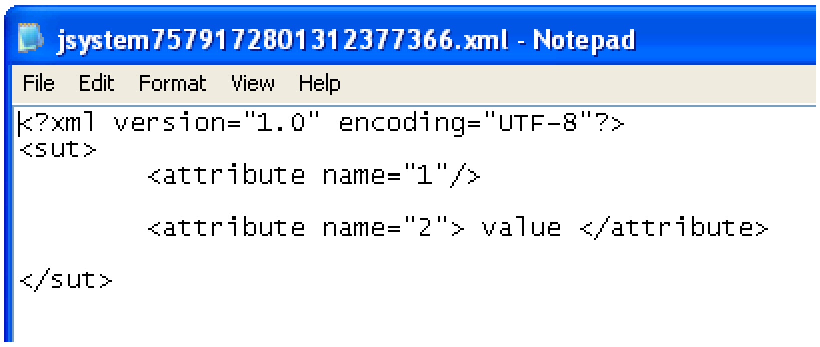
As you can see, a SutReader implementation has two methods called *getDocument*, one receiving the XML file and another receiving an input stream. Both methods return a Document object to be used by the JSystem framework.

Below is the original Sut file, before processing (after pressing the "Edit Sut" button):


jsystem8515704030307283731.xml - Notepad
File Edit Format View Help

```
<?xml version="1.0" encoding="UTF-8"?>
<sut>
        <attribute/>

        <attribute> value </attribute>

</sut>
```

And, using the ExampleReader, here is the processed Sut file (after pressing the “View Processed Sut” button), which will be used during JRunner runtime:


jsystem7579172801312377366.xml - Notepad
File Edit Format View Help

```
<?xml version="1.0" encoding="UTF-8"?>
<sut>
        <attribute name="1"/>

        <attribute name="2"> value </attribute>

</sut>
```

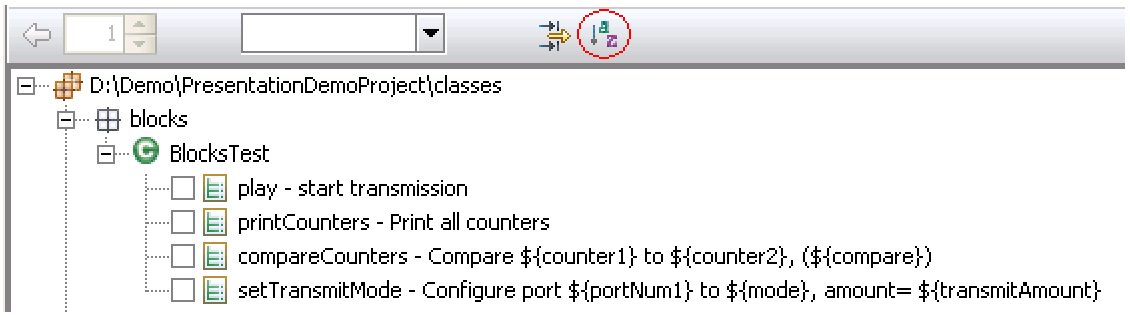
# 7. Assets Tree Customization

## 7.1 Alphabetical Sort

Until JSystem 5.6, the assets tree order was according to components class loading.

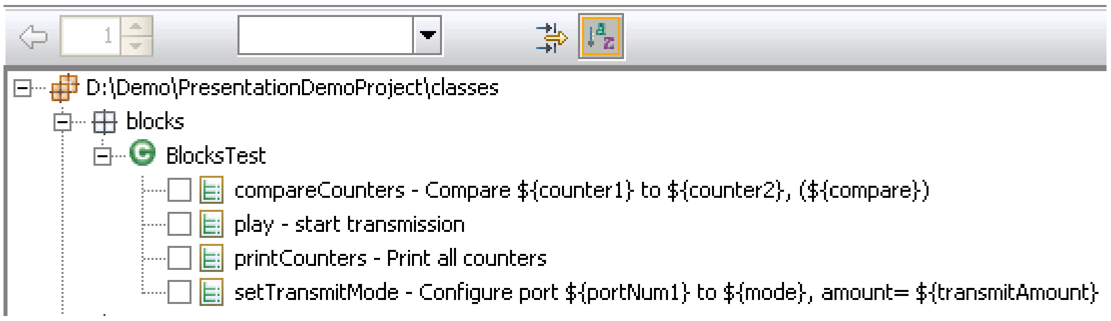
In JSystem 5.6 we have added the ability to sort the assets tree alphabetically.

In the following screen capture, the tests appear in the order they were loaded (default mode):



D:\Demo\PresentationDemoProject\classes
blocks
BlocksTest
play - start transmission
printCounters - Print all counters
compareCounters - Compare ${counter1} to ${counter2}, (${compare})
setTransmitMode - Configure port ${portNum1} to ${mode}, amount= ${transmitAmount}

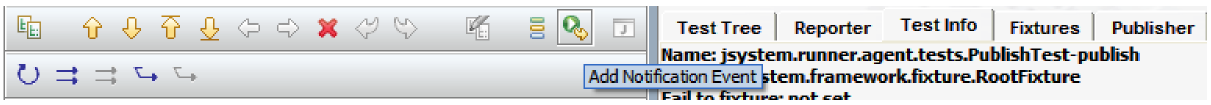
After pressing the “Sort Assets Tree Alphabetically” button, this is the new order:



D:\Demo\PresentationDemoProject\classes
blocks
BlocksTest
compareCounters - Compare ${counter1} to ${counter2}, (${compare})
play - start transmission
printCounters - Print all counters
setTransmitMode - Configure port ${portNum1} to ${mode}, amount= ${transmitAmount}

## 8. Publish Event

In this release the Publish event was renamed to Notification event which better reflects its task.



Name: jsystem.runner.agent.tests.PublishTest-publish
stem.framework.fixture.RootFixture
Fail to fixture: not set

The Notification event allows the user to add a notification test to the scenario. The event supports three actions:

1. Publish report.
2. Send email.
3. Init (clear) reports.

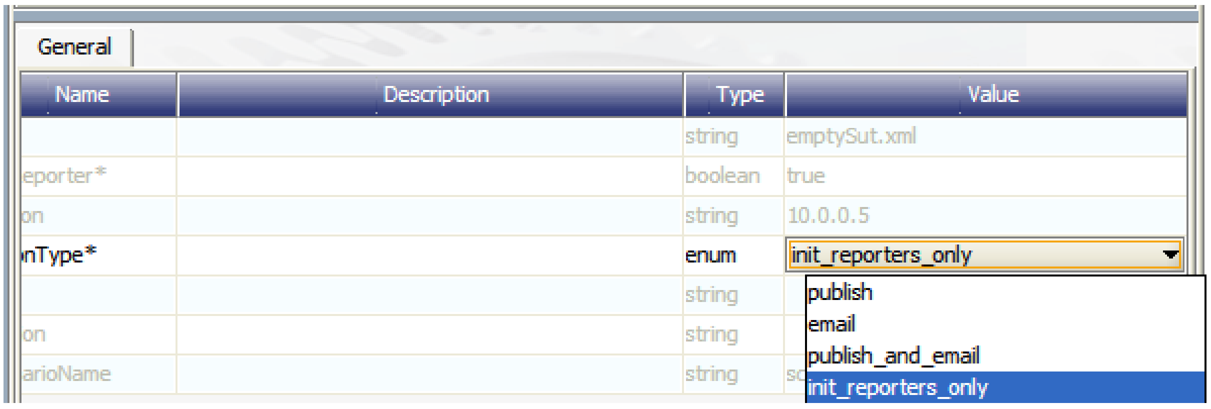
In previous JSystem releases the user could select one of the following combinations:

1. Publish
2. Email
3. Publish and email

In addition, the user could select whether to init the reporters or not.

In JSystem 5.6, in addition to the previous combinations, the user has the option to init the reporters without publishing or sending email.

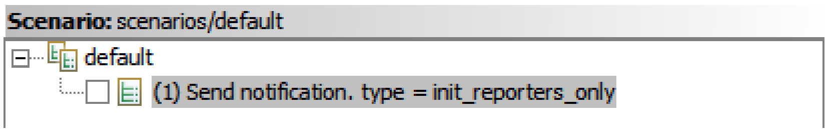
To support this action, a new entry was added to the ActionType parameter of the test.



| Name | Description | Type | Value |
|---|---|---|---|
| | | string | emptySut.xml |
| eporter* | | boolean | true |
| on | | string | 10.0.0.5 |
| nType* | | enum | init_reporters_only |
| | | string | |
| on | | string | |
| arioName | | string | sc |

Note that when selection init_reporters_only, all other parameters are not relevant and are grayed out.

Once the user sets the required action the test in the scenario tree will display the selected event type.



**Scenario:** scenarios/default

default
(1) Send notification. type = init_reporters_only

# 9. JSystem Core

## 9.1 Set System Object Exit Time

In JSystem 5.6 we have added the option to set the exit timeout for the system objects.

When JSystem closes all system objects, each system object close operation is run in its own thread. In previous JSystem releases, the main thread was waiting 1 second for each thread to complete. This 1 second timeout was constant and the system object had no control over it.

In JSystem 5.6 the system object can set the timeout request. This is useful for system objects that their close() method takes more than 1 second.

To set the timeout, the system object should simply call setExitTimeout(long) with the requested timeout in milliseconds.

## 9.2 Control System Object Reports

In JSystem 5.6 we have added an option for system object to let tests to control the amount of report messages the system object sends to the reporter.

In order to allow test to control the amount of messages, the system object should use SysteObjectImpl.report instead of report.report. These methods simply test whether the flag printStatuses is true or false before calling report.report. This flag is true by default, so all messages are reported, and the test can change it by calling the system object setPrintStatuses(Boolean) at any time during test execution.

The system objects can of-course, call report.report and bypass the test setting, forcing report messages.

# 10. Multiple Scenarios Execution

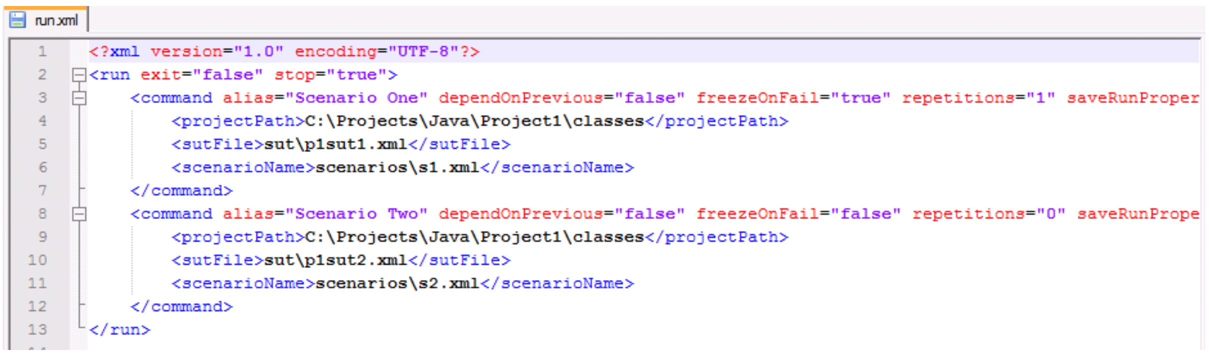
In order to reduce complex structures held in JSystem UI, a new mechanism was added to support several scenarios execution.

There are two options to execute multiple scenarios:
1) Using the new Multiple Scenarios Execution dialog.
2) Using the command line interface.

**Usage:**

- Create a multiple scenario execution flow using the Multiple Scenarios Execution dialog. The result of such an operation will be a creation of an instructions .xml file located in a folder of your choice.
  For example c:\jsystem\runner\**run.xml** or c:\temp\**execute.xml**
- Execute JSystem using command line new interface with a command line argument specifying the full path to the instruction .xml file.

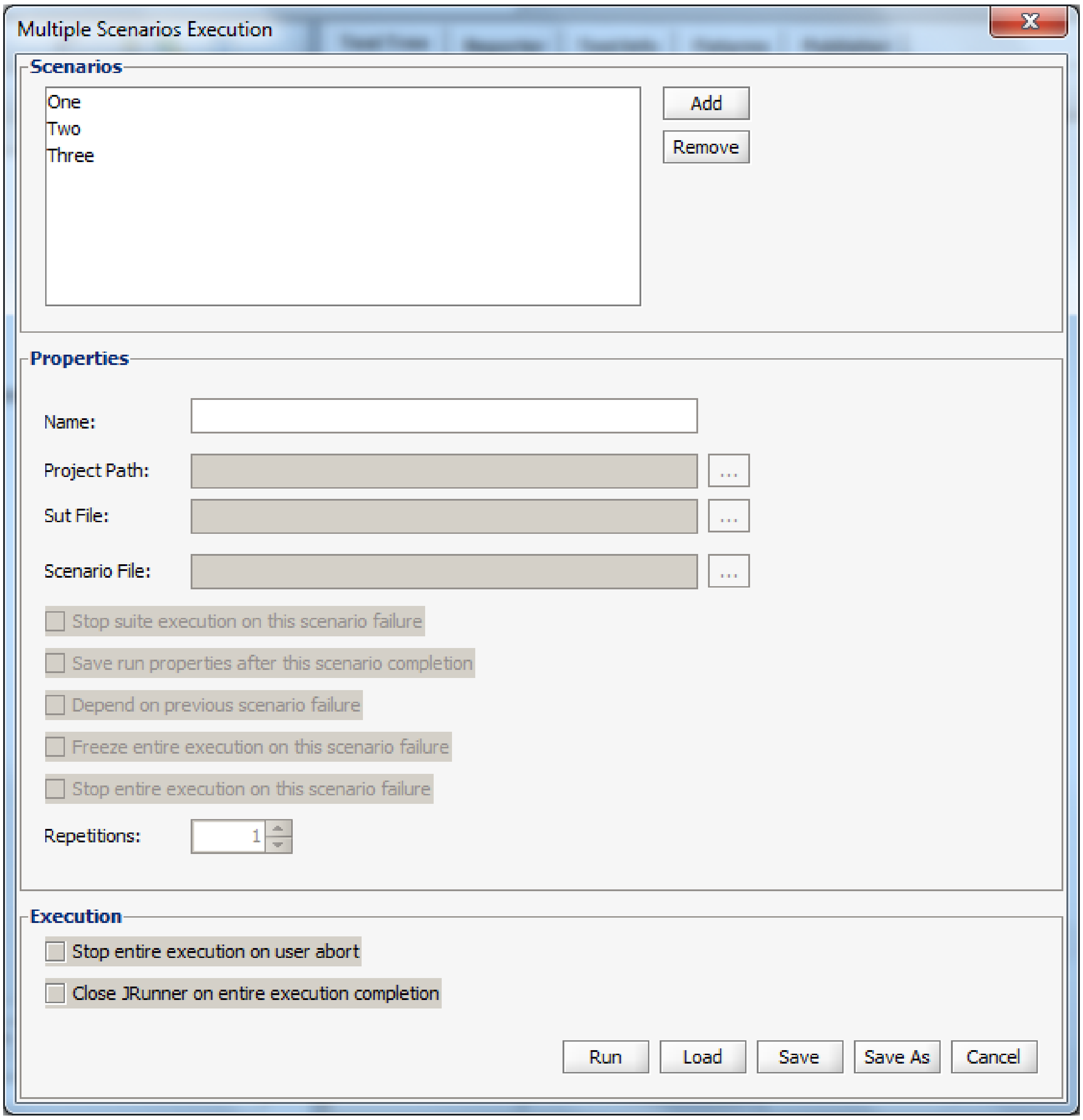
Below is an example of such a file:



```xml
<?xml version="1.0" encoding="UTF-8"?>
<run exit="false" stop="true">
    <command alias="Scenario One" dependOnPrevious="false" freezeOnFail="true" repetitions="1" saveRunProper
        <projectPath>C:\Projects\Java\Project1\classes</projectPath>
        <sutFile>sut\p1sut1.xml</sutFile>
        <scenarioName>scenarios\s1.xml</scenarioName>
    </command>
    <command alias="Scenario Two" dependOnPrevious="false" freezeOnFail="false" repetitions="0" saveRunPrope
        <projectPath>C:\Projects\Java\Project1\classes</projectPath>
        <sutFile>sut\p1sut2.xml</sutFile>
        <scenarioName>scenarios\s2.xml</scenarioName>
    </command>
</run>
```

## 10.1 Multiple Scenarios Dialog

In order to open the dialog, go to JSystem menu -> Tools -> Multiple Scenarios Execution. The following dialog will be open:



One
Two
Three

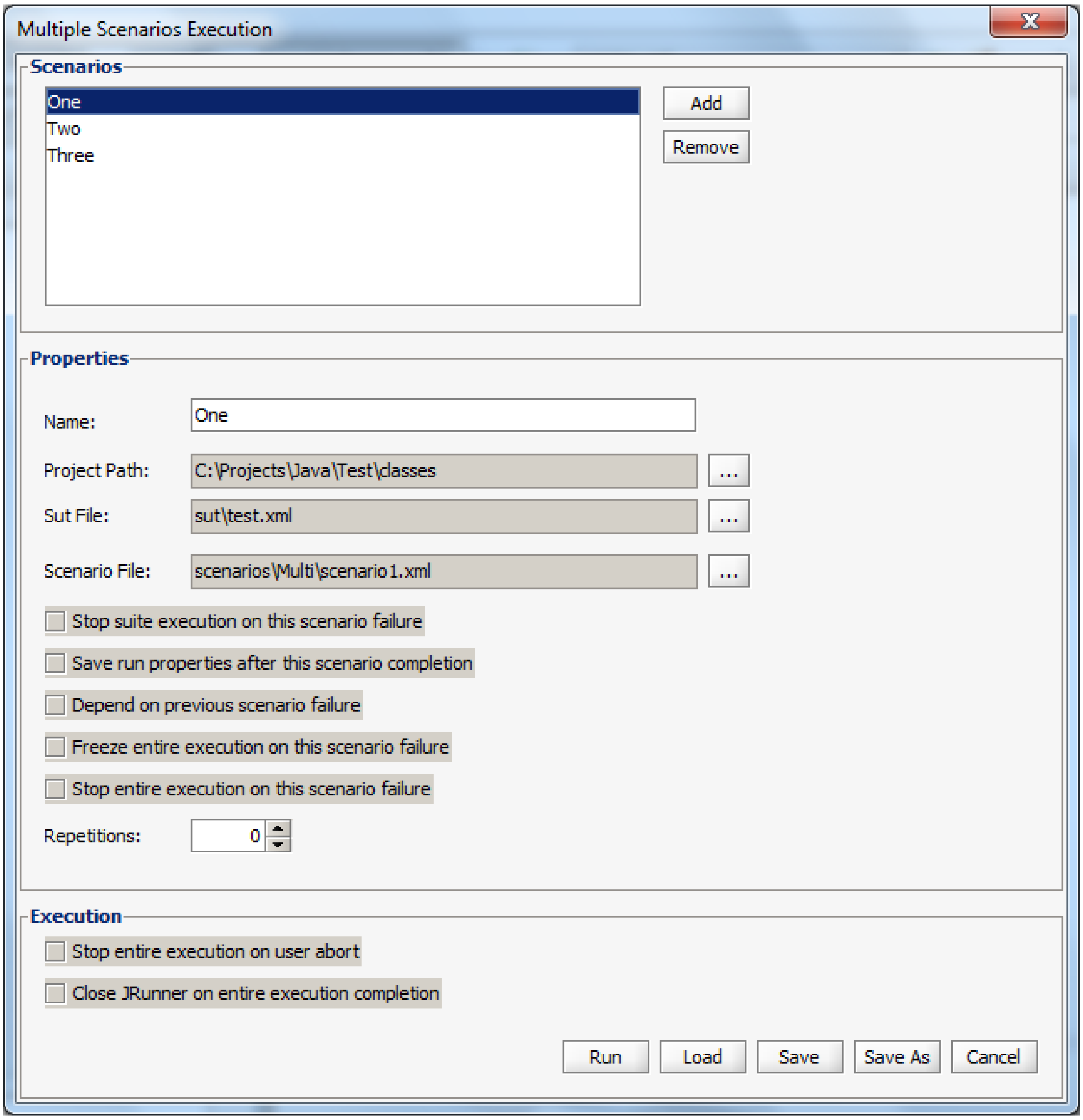
Name:
Project Path:
Sut File:
Scenario File:

- [ ] Stop suite execution on this scenario failure
- [ ] Save run properties after this scenario completion
- [ ] Depend on previous scenario failure
- [ ] Freeze entire execution on this scenario failure
- [ ] Stop entire execution on this scenario failure

Repetitions: 1

- [ ] Stop entire execution on user abort
- [ ] Close JRunner on entire execution completion

The dialog contains a list of scenarios and their properties. Once a new scenario is created or an already existing scenario is chosen the following properties are available:



Multiple Scenarios Execution

**Scenarios**

One
Two
Three

**Properties**

| | |
|---|---|
| Name: | One |
| Project Path: | C:\Projects\Java\Test\classes |
| Sut File: | sut\test.xml |
| Scenario File: | scenarios\Multi\scenario1.xml |

- [ ] Stop suite execution on this scenario failure
- [ ] Save run properties after this scenario completion
- [ ] Depend on previous scenario failure
- [ ] Freeze entire execution on this scenario failure
- [ ] Stop entire execution on this scenario failure

Repetitions: 0

**Execution**

- [ ] Stop entire execution on user abort
- [ ] Close JRunner on entire execution completion

### 10.1.1 Properties

**Name –** this is the alias name that will be stored in the configuration file. The scenario name is used for convenience only.

**Project Path** – this is the full path to the location of the project classes directory. This is the same location used when switching a project, using the "Switch Project" button.

For example if the project is located under c:\Projects\Automation\MyProject, then the following location should be chosen: c:\Projects\Automation\MyProject\classes.

**Sut File** – the name of the Sut file. The file chooser will automatically point to the location of available Sut files.

**Scenario File** – the name of the scenario file. The file chooser will automatically point to the location of available scenario files.

**Stop suite execution on this scenario failure** – when this option is set, in case there is a failure in one of the tests, it will stop execution of other tests located in this scenario.

**Save run properties after this scenario completion** – by default between each execution of JSystem, run properties are initialized. In case you want to keep the run properties between two consequent scenarios, mark this option.

**Depend on previous scenario failure** – in case this option is set, and previous scenario had errors, it will not allow execution of this scenario. Use this option when previous scenario is used as a "setup".

**Freeze entire execution on this scenario failure** – when this option is marked, if there is a failure in one of the tests of this scenario, it will freeze the whole execution. This option should be used mainly for debugging.

**Stop entire execution on this scenario failure** – when this option is marked, if there is a failure in one of the tests of this scenario, it will stop the whole execution.

### 10.1.2 Execution

**Note: All options listed in this section are general for the entire execution and not for a specific scenario.**

**Stop entire execution on user abort** – this option indicates what behavior should occur when the user manually presses the stop button in JSystem GUI.

Unmarked - next scenario execution will continue.

Marked – entire execution will be terminated.

**Close JRunner on entire execution completion** – this option indicates whether the JSystem should close after all scenarios are executed.

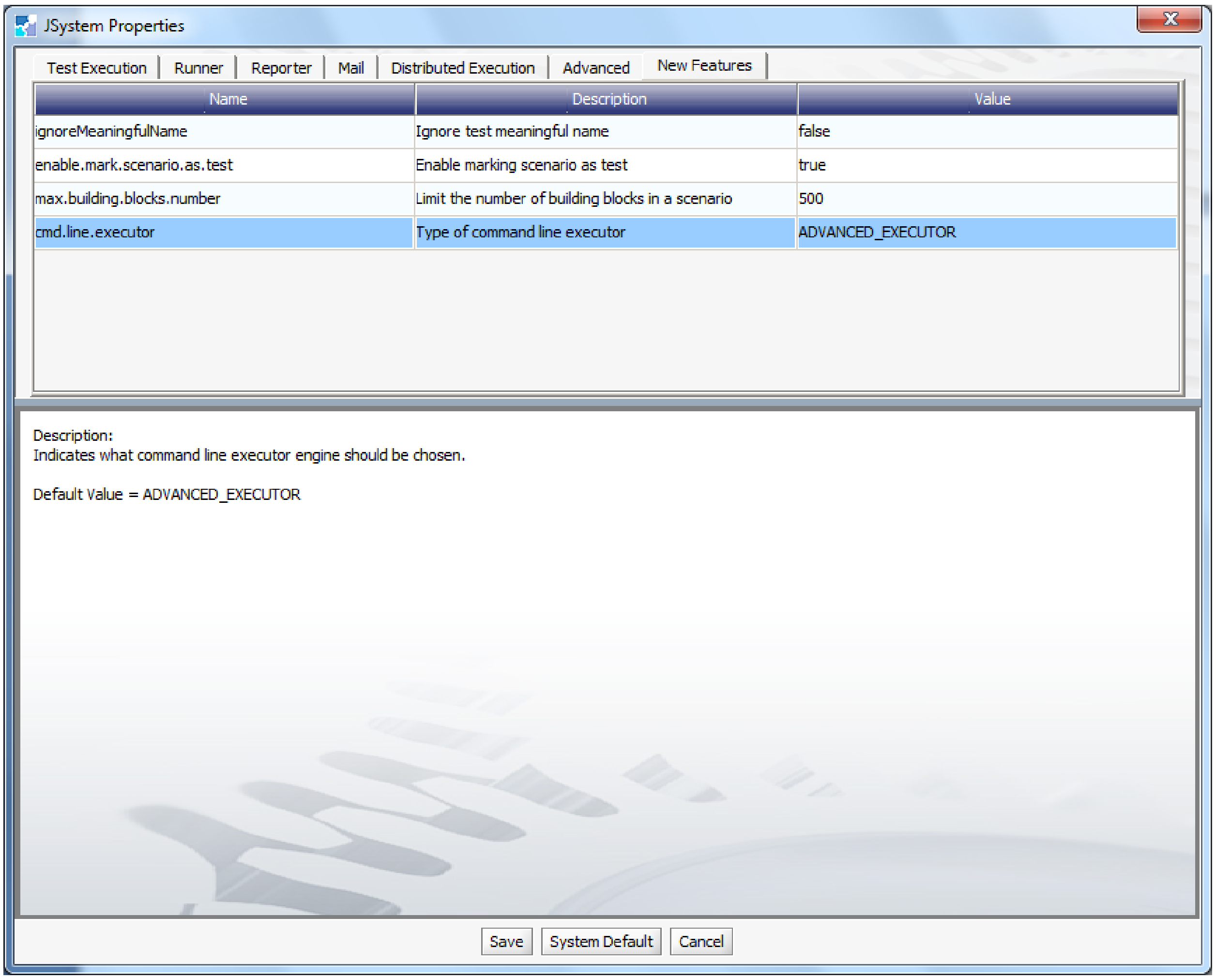
### 10.1.3 Control

In addition to the creation of the command line file, there is an option to load existing command line instruction files. This is done by pressing the "Load" button. Files can also be saved with different names using "Save As". Once the file is created it is recommended to test it by pressing the "Run" button.

## 10.2 Command line execution

Once the multiple scenarios execution instruction file has been created, it can be executed using JSystem command line interface.

To enable JSystem command line interface to work with multiple scenarios go to JSystem menu -> Tools -> JSystem Properties. Once JSystem Properties dialog is open choose the "New Features" tab.



| Name | Description | Value |
|---|---|---|
| ignoreMeaningfulName | Ignore test meaningful name | false |
| enable.mark.scenario.as.test | Enable marking scenario as test | true |
| max.building.blocks.number | Limit the number of building blocks in a scenario | 500 |
| cmd.line.executor | Type of command line executor | ADVANCED_EXECUTOR |

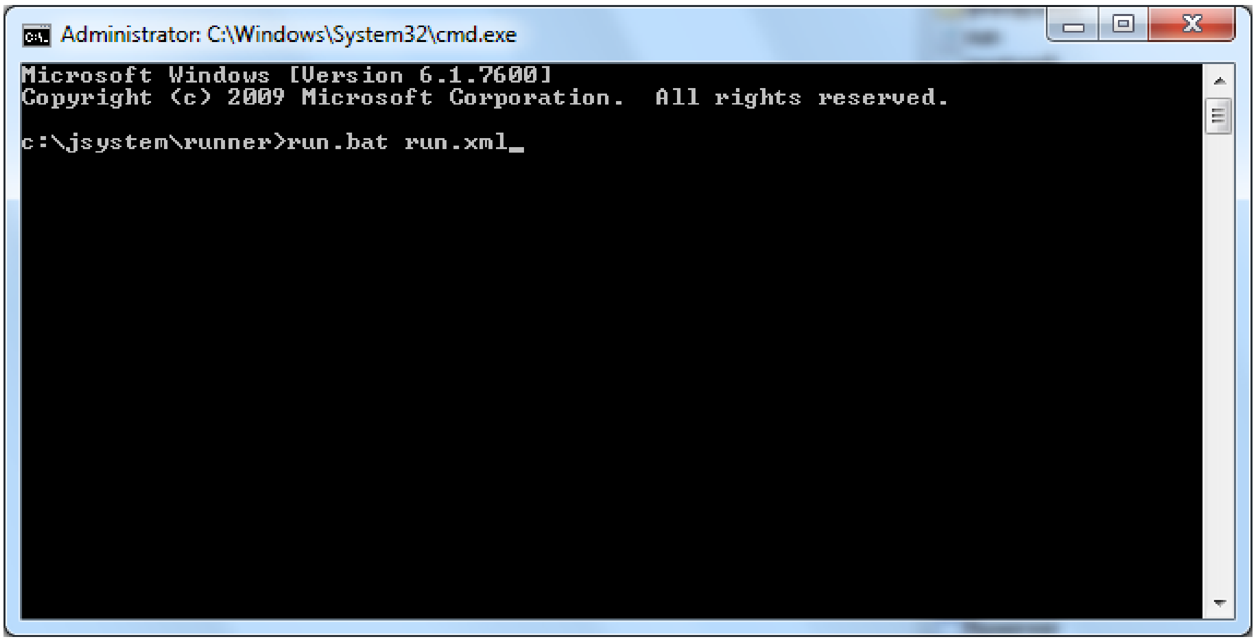
Description:
Indicates what command line executor engine should be chosen.

Default Value = ADVANCED_EXECUTOR

There is a new option called "cmd.line.executor". Please make sure that it's configured to "ADVANCED_EXECUTOR" option. Click the "Save" button to make sure that changes are saved.

Once JSystem command line interface is configured, use the following syntax from the command line to run JSystem with multiple scenarios:

run.bat <name_of_instruction_file.xml>



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

c:\jsystem\runner>run.bat run.xml_
```

After the above commands are executed, the JSystem will open and will start executing scenarios according to the instruction file.