

Chapter 11

JSystem Automation Framework Quick Start Project



> In this chapter...

<i>Quick Start Project Overview</i>	<i>Page 2</i>
<i>Step 1: Enable the CLI Agent</i>	<i>Page 3</i>
<i>Step 2: Creating a New Eclipse Workspace</i>	<i>Page 3</i>
<i>Step 3: Importing the JRunner into the Workspace</i>	<i>Page 3</i>
<i>Step 4: Creating a New SystemObject Project</i>	<i>Page 5</i>
<i>Step 5: Creating Tests Project</i>	<i>Page 8</i>
<i>Step 6: Writing a Ping Test and Scenario</i>	<i>Page 11</i>
<i>Step 7: Adding Test Parameters</i>	<i>Page 18</i>
<i>Step 8: Writing the System Object Skeleton</i>	<i>Page 20</i>
<i>Step 9: Implementing the Ping Command</i>	<i>Page 35</i>
<i>Step 10: Performing Analysis</i>	<i>Page 42</i>
<i>Step 11: Writing the Fixtures</i>	<i>Page 47</i>
<i>Implementation of a Fixture</i>	<i>Page 49</i>
<i>Associating a test with a Fixture</i>	<i>Page 52</i>

11.1 Quick Start Project Overview

Welcome to the Quick Start project. In this tutorial we will walk through the process of writing a small project that installs a patch on a device under test (DUT), alter its hosts table and verifies that the machine has a ping to a remote address. (Note that not all operations will be actually implemented, just parts which are necessary to understand JSystem)

Note: The operations are activated using the JSystem connectivity package (telnet/SSH). You will therefore be able to activate the tests on either the local machine or any other machine in the setup.

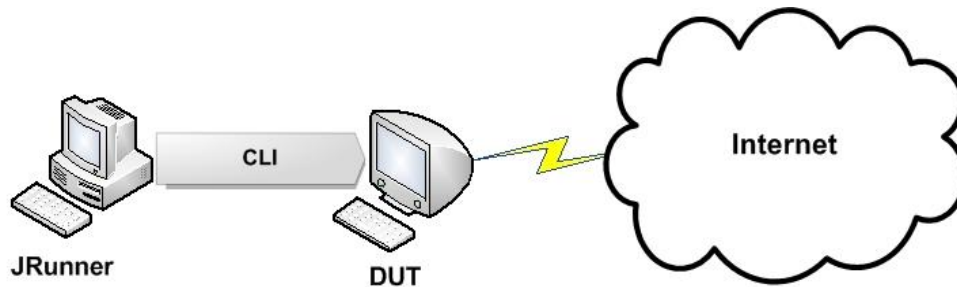


Figure 1: Quick Start Illustration

The example setup includes two machines, the first is installed with the JRunner, the second is the device under test (DUT), in this test case the DUT is another windows/linux machine.

11.1.1 Tutorial steps

We will start with enabling CLI agent on the DUT (telnet/ssh), setting up Eclipse workspace (I assume that at this stage JSystem is installed), and creating two Eclipse projects - the system object and tests projects. We will continue with writing the skeleton of the first test in the scenario. After completing test skeleton you will learn how to extend the functionality of the test enabling it to get parameters from the JRunner user.

Once the skeleton of the scenario is ready you will be able to add a skeleton of the system object which models the DUT. After completion, implement the **"ping command"** using the CLI package.

We will pure content to the test by incorporate the system object in it and analyzing ping command results.

At the end of the project, you will add two fictitious fixtures, the first deploys the patch and second one alters the hosts table. You will then incorporate these fixtures in the scenario.

11.2 Step 1: Enable the CLI Agent

Before starting to write the project, you need to make sure that CLI agent (Telnet on windows or SSH on Linux) is enabled on the DUT.

Reference: Chapter 15 Enabling the CLI Agent

The CLI agent must be enabled on the machine being tested. The CLI agent on Windows is Telnet and SSH on Linux.

Enabling Telnet on Windows XP/2003

http://www.petri.co.il/enable_telnet_on_windows_2003.htm

Enabling Telnet on Vista

<http://windowshelp.microsoft.com/Windows/en-US/Help/81b6d4b7-905e-4d70-8379-7934913fedb01033.mspx>

SSH Server on Linux

If you are using a version of Linux that was released after 2002, you already have OpenSSH installed. Here are some links which explain SSH and how to solve common problems:

[SSh tutorial](#)

[Secure Copy and Login](#)

[SSh on fedora](#)

11.3 Step 2: Creating a New Eclipse Workspace

The second step is to create a new Eclipse workspace.

To create a new workspace:

1. Start Eclipse.
2. Select **File > Switch Workspace**. The Select a workspace window appears.

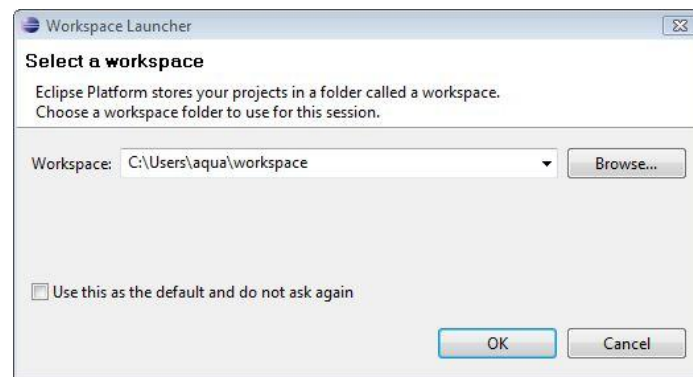


Figure 2: Workspace Launcher

3. Enter the path to the new workspace and then click OK (Eclipse will create a new folder if the folder does not exist).

11.4 Step 3: Importing the JRunner into the Workspace

The next step is to import JRunner into the new workspace.

Importing a JRunner into the workspace:

1. In Eclipse, select **File > "Import"**. The Import window opens.

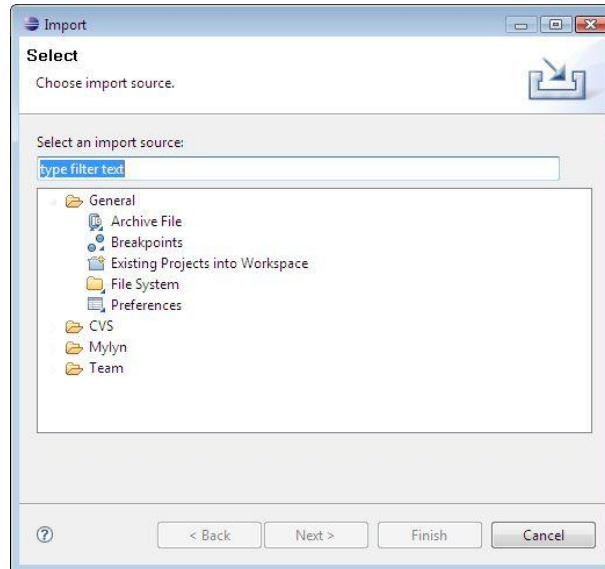


Figure 3: JSystem Import Window

2. Select **"Existing Projects into Workspace"** and click **"Next"**. The **"Select a directory to search for existing Eclipse projects"** window opens.
3. Click **"Browse"**. The **"Browse for Folder"** dialog opens.

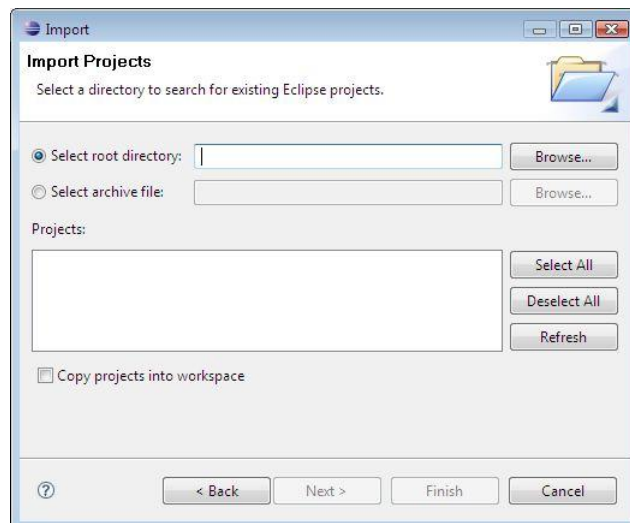


Figure 4: Import Projects and Browse for Folder Window

4. Select **“runner”** folder and click **“OK”**.

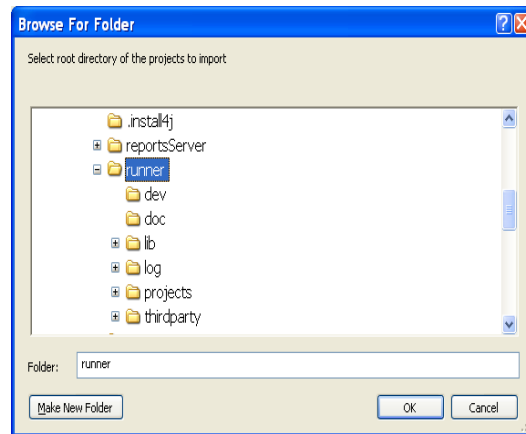


Figure 5: Browse For Folder Window

5. Select **“runner”** project and click **“OK”**.

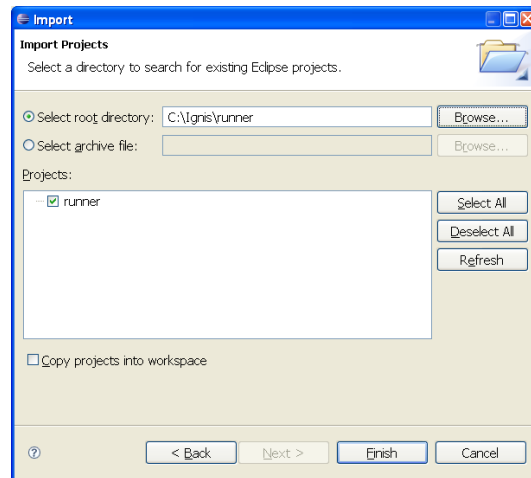


Figure 6: Select runner

6. Click **“Finish”**.

11.5 Step 4: Creating a New SystemObject Project

The first layer in JSystem architecture is the SystemObjects layer. In our example we will have one SystemObject which manages/represents the windows/linux machine that we are testing.

In order to create a new System Object project, perform the following steps.

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

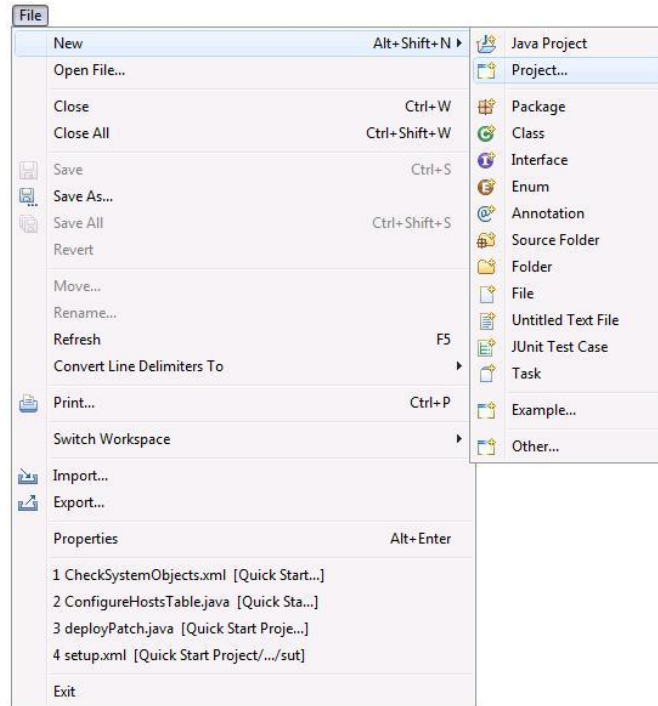


Figure 7: Creating a New Test Project

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

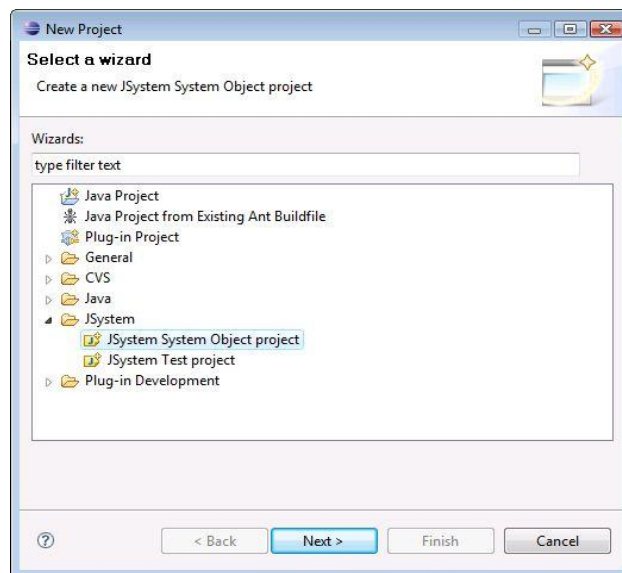


Figure 8: JSystem New Project Wizard

3. In the New project dialog, you can see the list of System Objects available. Select the **"CLI"** package, give your project the name **"MyStation"** and press **"Finish"**.

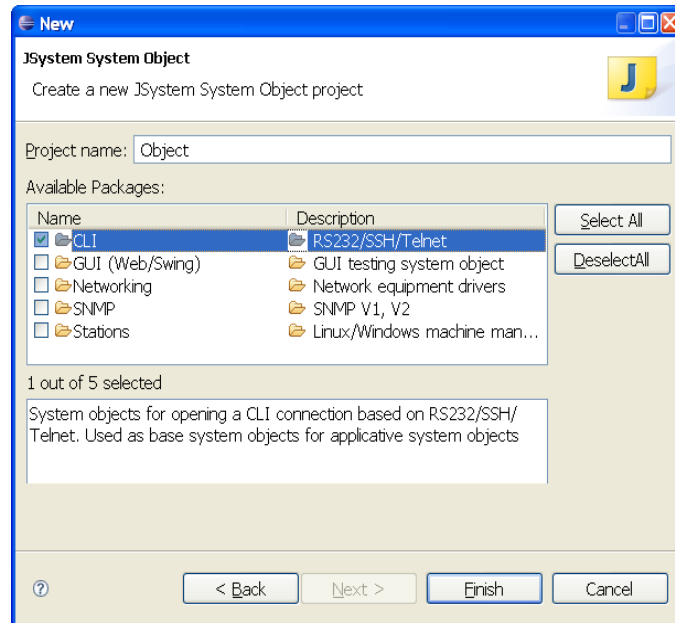


Figure 9: JSystem New Project Wizard 2

4. A System Object project with all relevant folders is created.

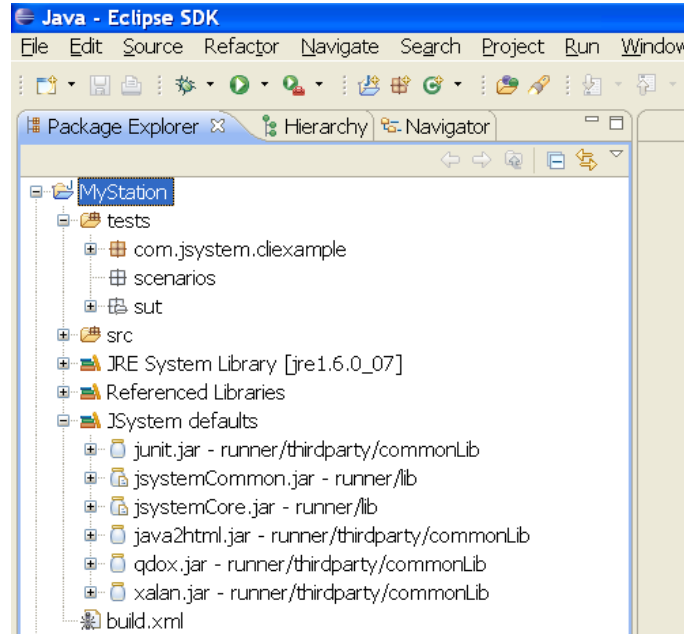


Figure 10: System object project structure

Let's review the system object project structure and files:

File/Folder	Description
src	System object source folder.
test	System object unit tests folder. Will not be used in this project.
com.jsystem.cliexample	Example code that comes with the CLI package.
JSystem defaults	Bundle of jars that are mandatory for JSystem projects
build.xml	Ant build file for compiling, packing and distributing system object to the tests project.

11.6 Step 5: Creating Tests Project

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

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

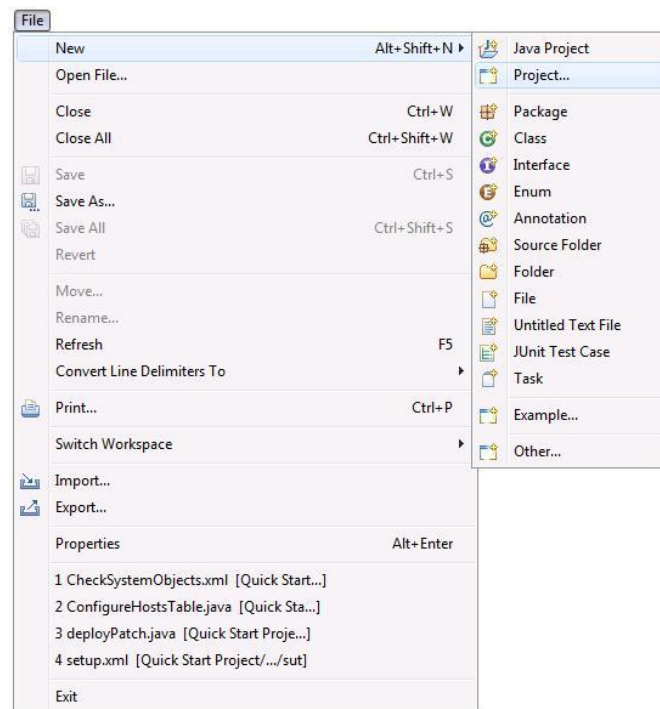


Figure 11: Creating a New Test Project

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

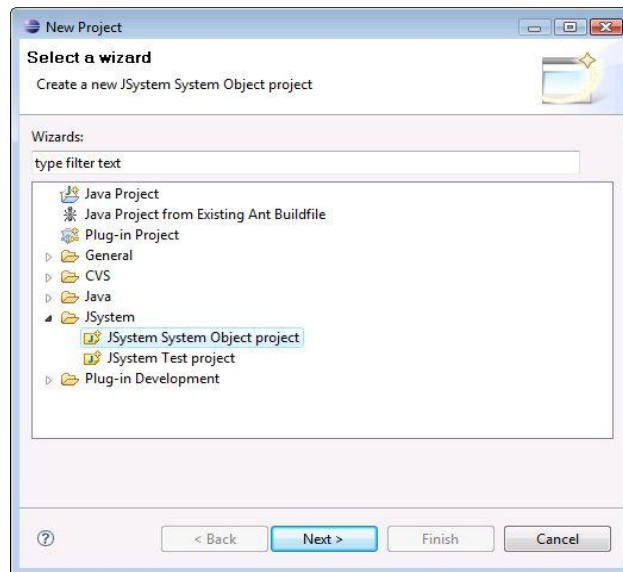


Figure 12: JSystem New Project Wizard

3. In the New project dialog, you can see the list of System Objects available, leave the list un-checked, give your project the name **"QuickStart"** and press **"Finish"**.

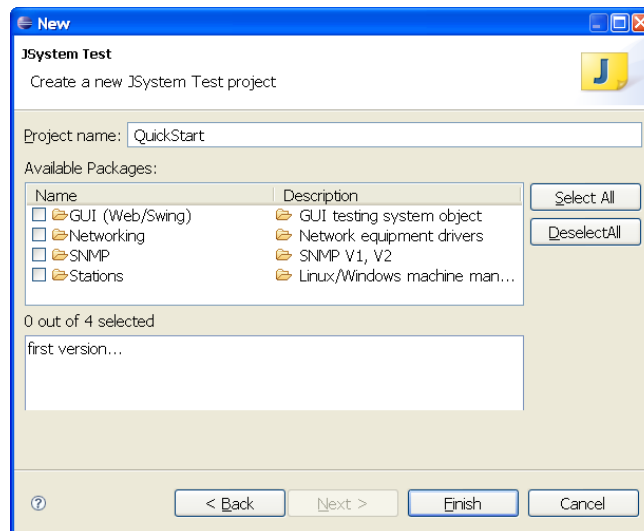


Figure 13: JSystem New Project Wizard 2

4. A Tests project with all relevant folders is created:



Figure 14: Tests project structure

File/Folder	Description
Src	In tests projects this source folder is usually not used.
Test	Tests source folder. Includes two more packages: sut package for sut files (you will learn about it later on) and scenarios package for jsystem scenarios.
JSystem defaults	Bundle of jars that are mandatory for JSystem projects
Lib	Workspace's system object jars should be distributed to this folder. When the JRunner loads the project, it also loads all the jars in the lib folder. The build.xml file of the system object does the distribution of the system object jar to this folder. We will see it later on.
resources	If your tests require additional resources like property files, scripts, images or any other resource this is where these resources should reside. The JRunner packs this folder with the other folders of the project when working in a distributed setup.

Why do we create two projects? Separation between system object projects and test project allows easy reuse of system objects and keeps the workspace more clean and easy to maintain.

The benefits of this separation become clearer in big automation projects. In our case this separation is artificial, but still I want you to work according to JSystem best practices.

11.7 Step 6: Writing a Ping Test and Scenario

The test will be written in the “**QuickStart**” test project.

First create a new java package for your test. Name the package “**org.jsystem.quickstart**”.

11.7.1 Creating a New Package

1. In the package explorer tab, “**right click**” on the tests folder and select “**new→Package**”.

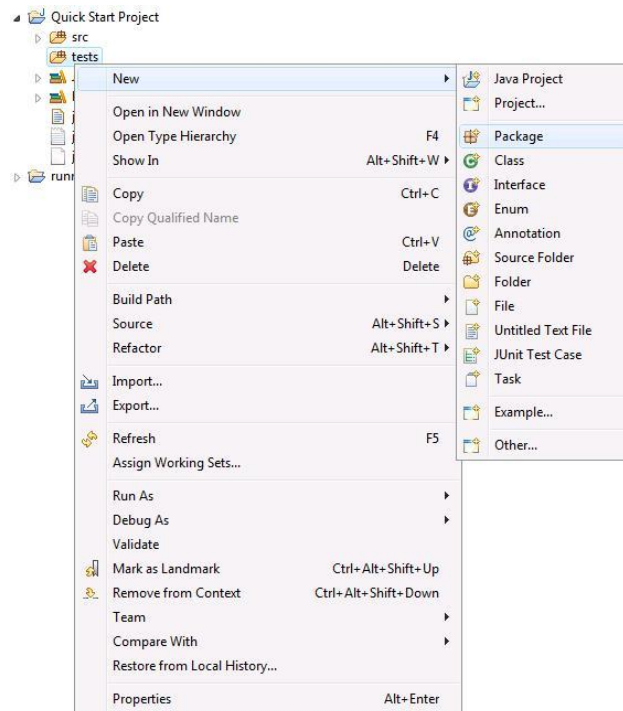


Figure 15: Creating a New Package

2. In the dialog input field enter “**org.jsystem.quickstart**”.

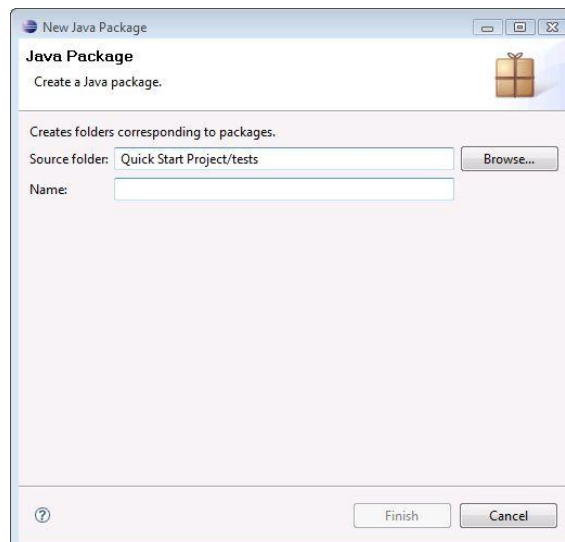


Figure 16: Entering package Information

3. Press **"Finish"**.

11.7.2 Creating test class

1. Now let's create the test class, we will name the test class **"HostSanityTest"**.
2. **"Right click"** on the **"org.jsystem.quickstart"** package and select **"new→Class"**.

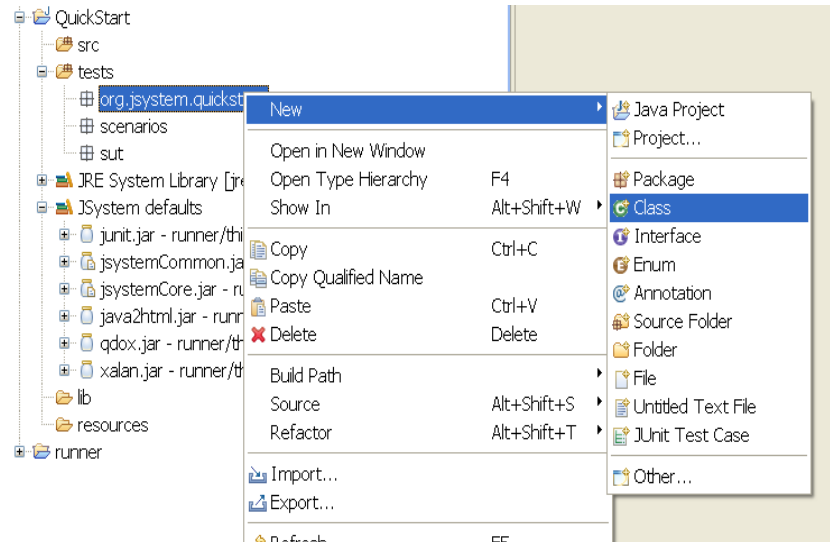


Figure 17: New Class

3. In the **"New Class"** dialog, **"Name"** field enter **"HostSanityTest"**
4. In the **"Superclass"** field write **"junit.framework.TestCase"**

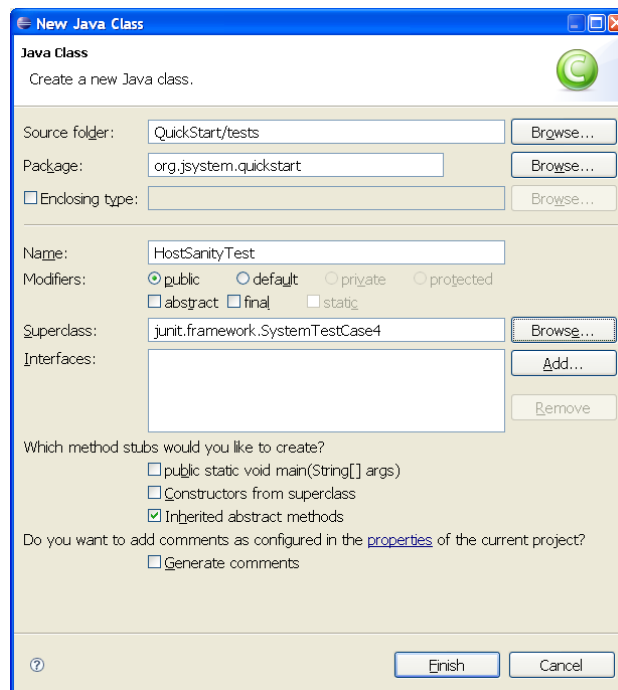


Figure 18: New Class Window

5. Press **"Finish"**.

- Next, add your test case method. The name of the method is **"pingFromDUT()"**.
- Don't forget to add the `@Test` junit4 annotation so the system will identify the method as a test

Here is the test class after adding the test case.

```
package org.jsystem.quickstart;

import org.junit.Test;

import junit.framework.TestCase4;

public class HostSanityTest extends SystemTestCase4 {

    @Test
    public void pingFromDUT() {

    }

}
```

Table 1: Test skeleton Code

11.7.3 Running the test

- Before you run the test from JRunner, run it from within eclipse. Select the test class in the package explorer view **"right click"** and select **"Run As—> JUnit Test"** as the output of the operation.

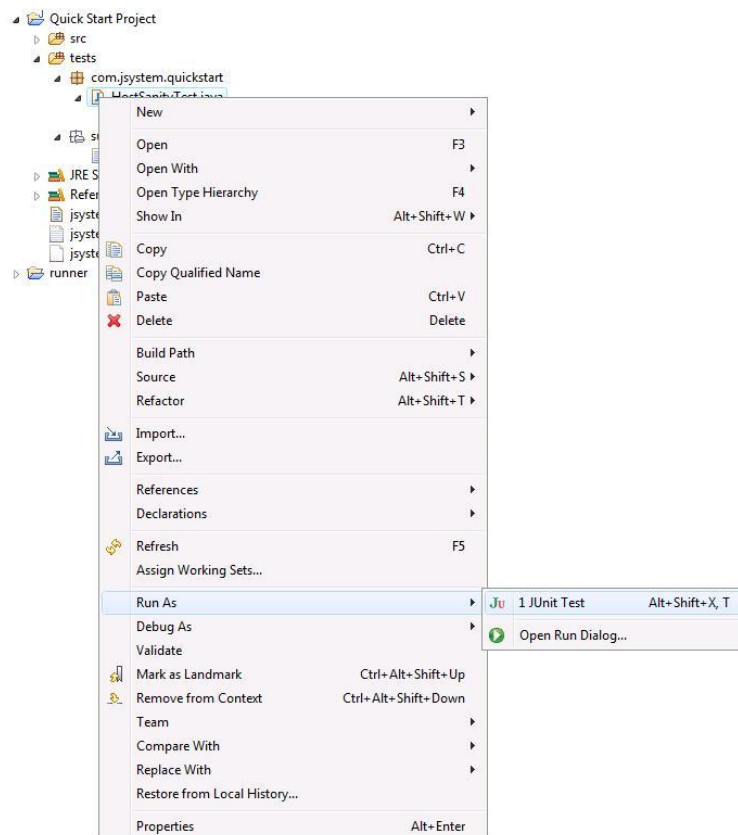
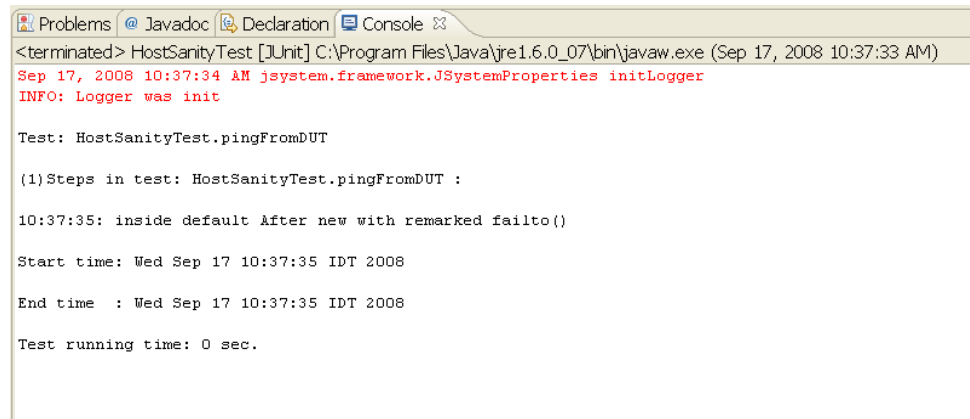


Figure 19: Junit Run Test

2. Run Test.



```
<terminated> HostSanityTest [JUnit] C:\Program Files\Java\jre1.6.0_07\bin\javaw.exe (Sep 17, 2008 10:37:33 AM)
Sep 17, 2008 10:37:34 AM jsystem.framework.JSystemProperties initLogger
INFO: Logger was init

Test: HostSanityTest.pingFromDUT

(1)Steps in test: HostSanityTest.pingFromDUT :

10:37:35: inside default After new with remarked failto()

Start time: Wed Sep 17 10:37:35 IDT 2008

End time : Wed Sep 17 10:37:35 IDT 2008

Test running time: 0 sec.
```

Figure 20: Console output

3. The Console opens and shows the test results.

11.7.4 Opening the JRunner

Now let's open the JSystem JRunner and run the test from within the JRunner application.

1. In the workspace go to the JRunner project and run the **"run.bat"** and activate it.

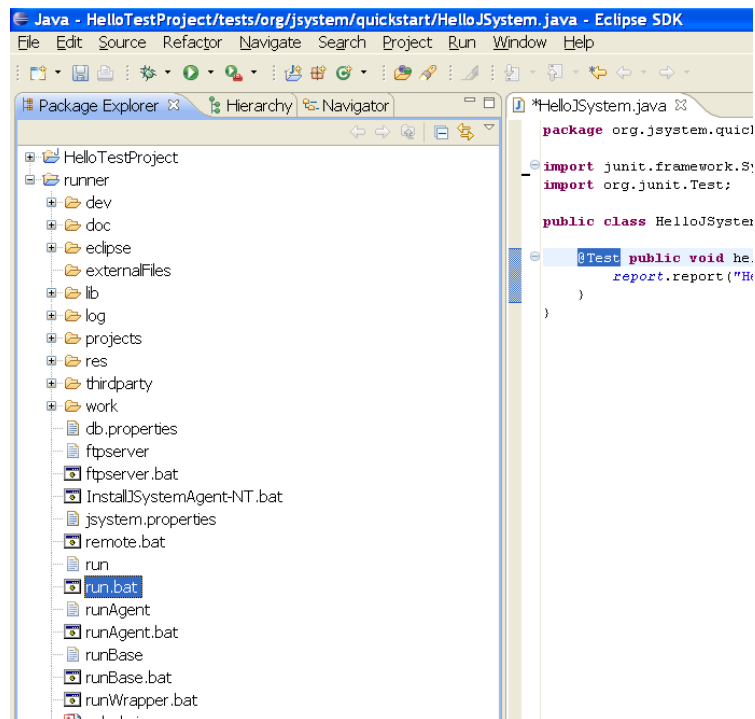


Figure 21: Activating the runner from eclipse

Notes:

- *If you are working on Linux, the JRunner can't be activated from Eclipse, open a shell console change directory to <runner> folder and activate the "run" script (./run)*
- *When running JRunner for the first time after the installation, jsystemSerices example project is opened.*

2. Press on the "File"→"Switch Project"

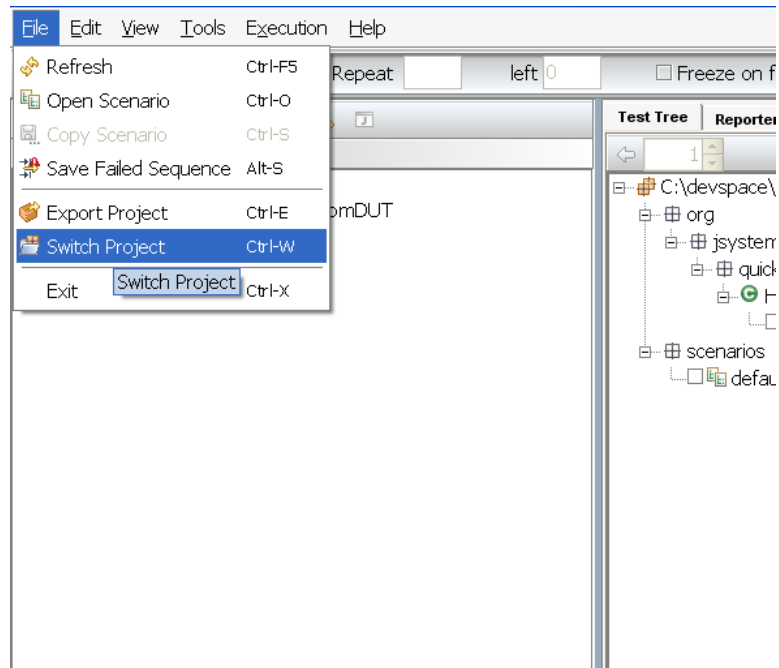


Figure 22: Switch project menu item

3. Browse to the "classes" folder of the "QuickStart" project

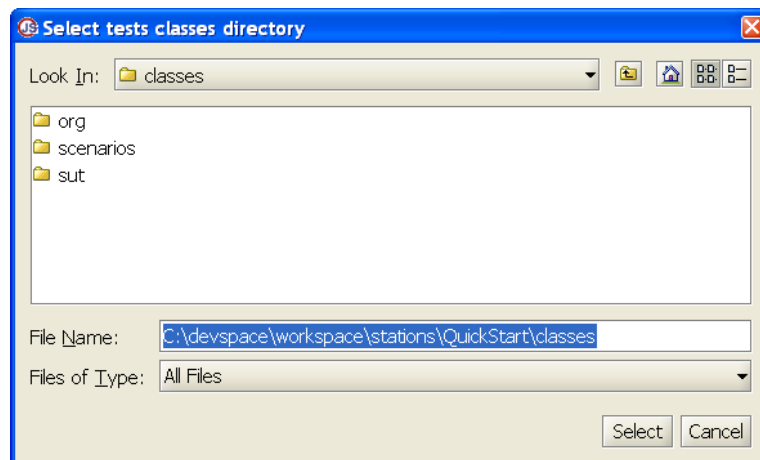


Figure 23: Select Test classes Directory

- Once the JSystem JRunner opens your newly created project, test will appear in the **"Test Tree"** tab control on the right side of the JSystem JRunner.

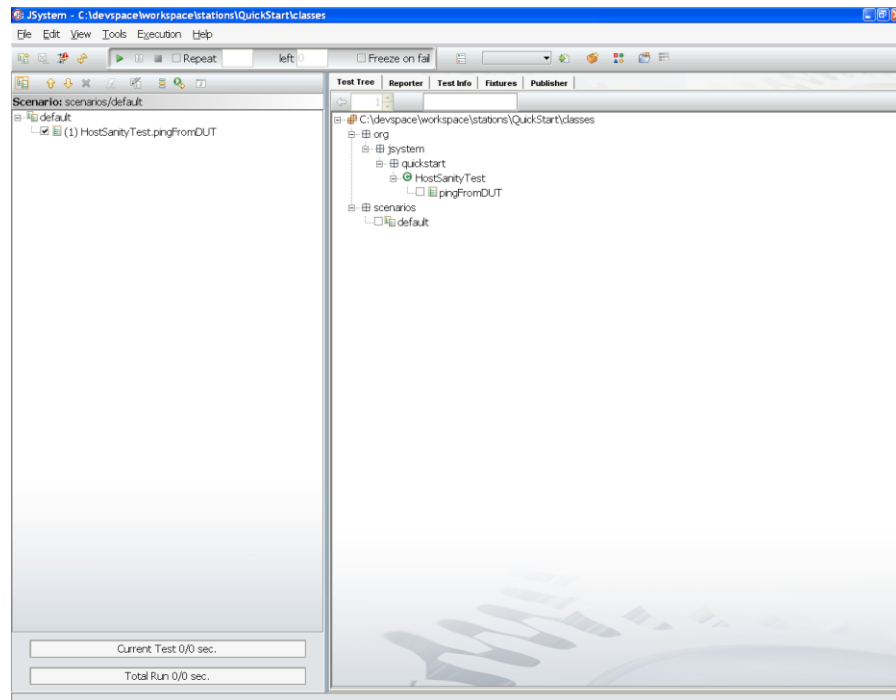




Figure 246: JSystem Main JRunner Window

- Now select the test and press the  **"Add tests"** arrow button to transfer the test into the default scenario which appears in the left hand side of the JSystem JRunner window.
- Now press the  **"Run Scenario"** button to run the scenario. Now select your project.

Once the scenario commences its progress can be viewed in the "Reporter" tab, highlighting the main execution messages.

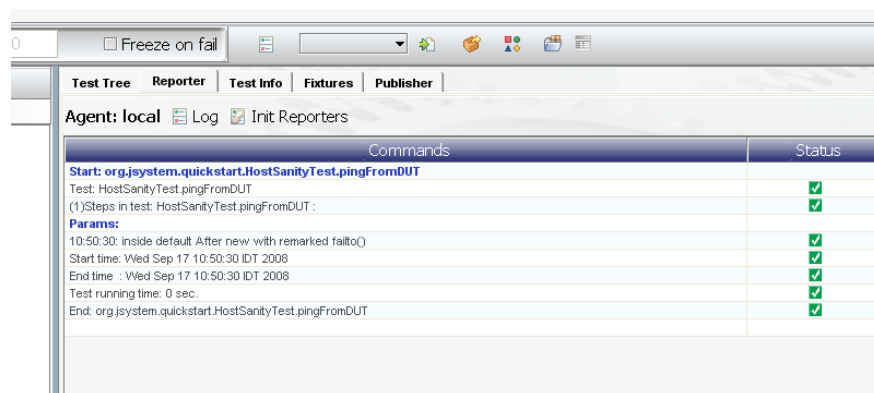



Figure 257: Reporter tab

7. In order to view the full execution results press the  "Log" button in order to open an HTML report page.

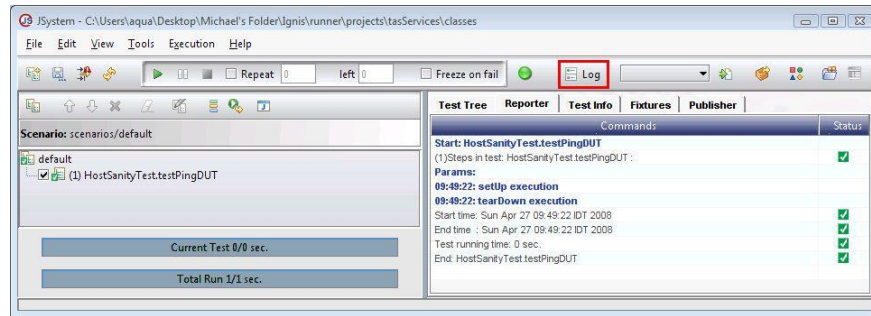


Figure 26: The Reporter (log) button

11.8 Step 7: Adding Test Parameters

Now that a test has been created and added to a scenario it's time to learn how to sophisticate the test case. In order to enable the JRunner user to determine the ping packet size, test class members with setters and getters must be implemented in the test.

```
package org.jsystem.quickstart;

import org.junit.Test;

import junit.framework.TestCase4;

public class HostSanityTest extends SystemTestCase4 {
    private String pingDestination;
    private int packetSize;

    @Test
    public void pingFromDUT() {

    }


    public String getPingDestination() {
        return pingDestination;
    }

    public void setPingDestination(String pingDestination) {
        this.pingDestination = pingDestination;
    }

    public int getPacketSize() {
        return packetSize;
    }

    public void setPacketSize(int packetSize) {
        this.packetSize = packetSize;
    }
}
```

Table 2: Adding Test Parameters

1. Once the code has been added to the test, press Ctrl-S to save the file.
2. Return to the JRunner and refresh our project by pressing the  **"Refresh"** button.

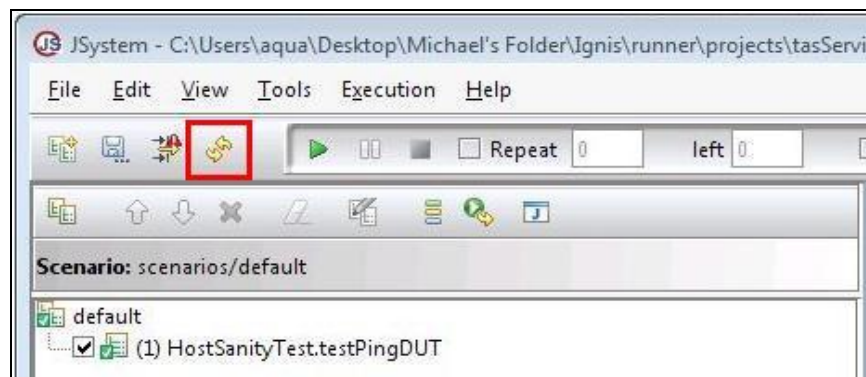
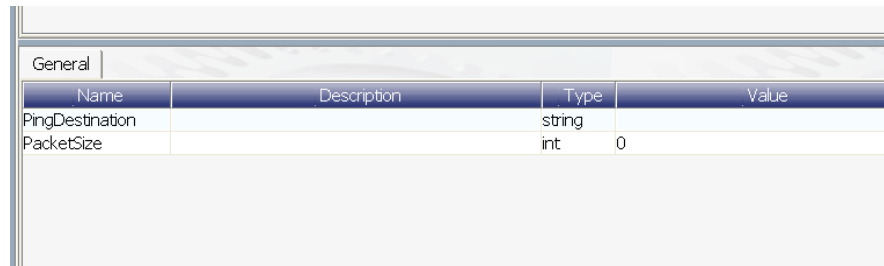


Figure 278: Refreshing the JRunner Window

- When refreshing the project, the JRunner reloads the compiled java code, by **"double clicking"** on the test in the scenario, the **"Test Info"** tab will be brought into focus and the test parameters that have been defined will be shown.

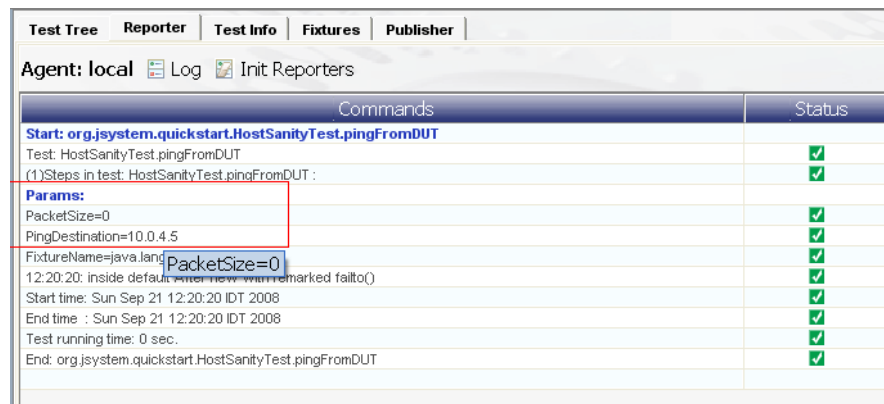


The screenshot shows the 'Test Info' tab with a 'General' sub-tab. It contains a table with test parameters.

Name	Description	Type	Value
PingDestination		string	
PacketSize		int	0

Figure 28: Test Parameters

Now let's give these two parameters values and run the scenario again, at these stage our test doesn't do anything with these parameters, but we can see the values of the parameters in the "Reporter" tab and in the log.



The screenshot shows the 'Reporter' tab with a table of test execution commands and their status. A red box highlights the 'Params' section, and a blue box highlights the 'PacketSize=0' parameter.

Commands	Status
Start: org.jsystem.quickstart.HostSanityTest.pingFromDUT	
Test: HostSanityTest.pingFromDUT	✓
(1)Steps in test: HostSanityTest.pingFromDUT :	✓
Params:	
PacketSize=0	✓
PingDestination=10.0.4.5	✓
FixtureName=java.lang	✓
12:20:20: inside default	✓
12:20:20: new	✓
12:20:20: remarked failto()	✓
Start time : Sun Sep 21 12:20:20 IDT 2008	✓
End time : Sun Sep 21 12:20:20 IDT 2008	✓
Test running time: 0 sec.	✓
End: org.jsystem.quickstart.HostSanityTest.pingFromDUT	✓

Figure 29: Parameters values as seen in reporter tab

Note: Exposing test parameters to the JRunner user is part of the "Multi user support" service.

11.9 Step 8: Writing the System Object Skeleton

Now that the test skeleton has been created it is time to begin filling it. The next step in the project is to write the **system object** that abstracts the work with the DUT.

We will start with writing the system object class skeleton, and then continue with incorporating the system object skeleton in the test, at the end of this section you will write the SUT xml file that defines the configuration of the system object.

11.9.1 Writing a System Object Class

Note: The system object will be written in the “MyStation” project.

1. Create a new java package for the system object. The JSystem convention is that system objects are written under the “**src**” source folder that is where the package should be created.

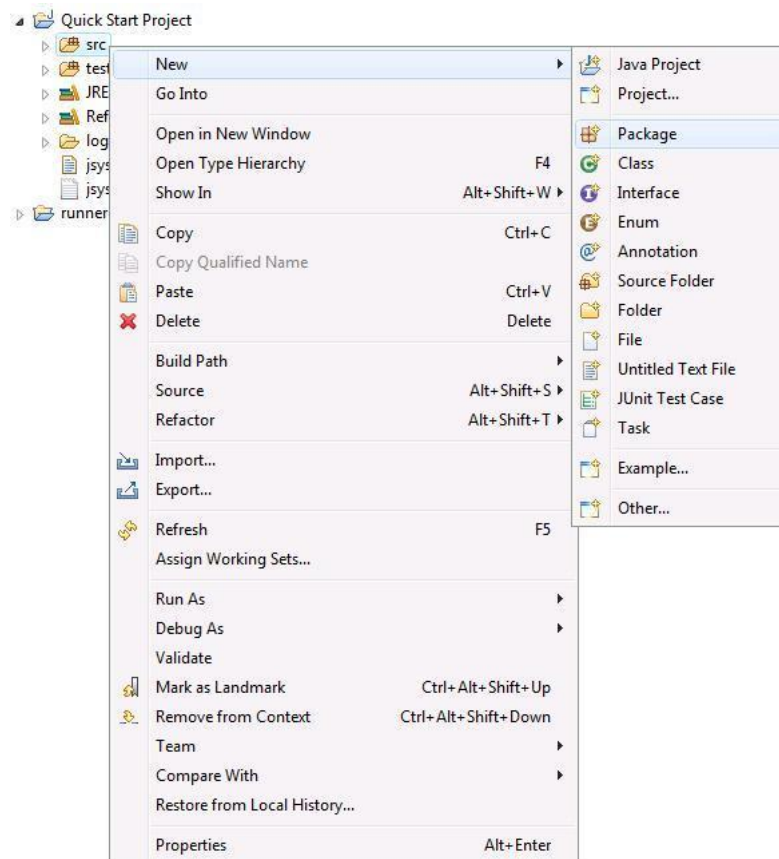


Figure 30: New Package Creation

2. Name the package **"org.jssystem.quickstart"**

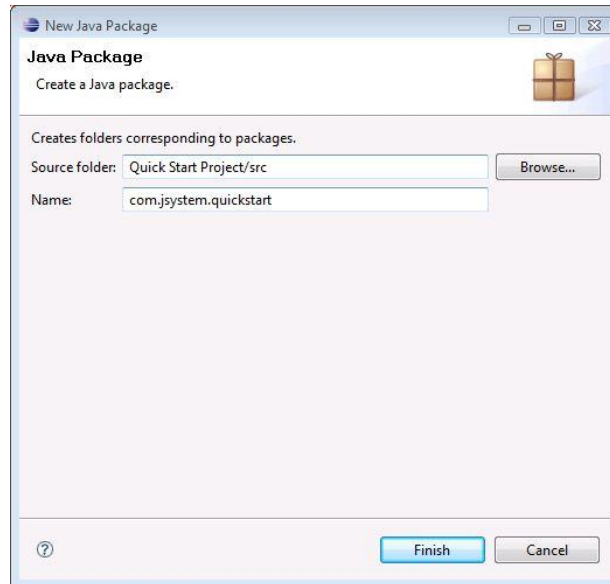


Figure 31: Naming the New Package

3. Click the **"Finish"** button.
4. Now create a new system object **"class"** file by right clicking on the newly created **"package"** and selecting a new class.

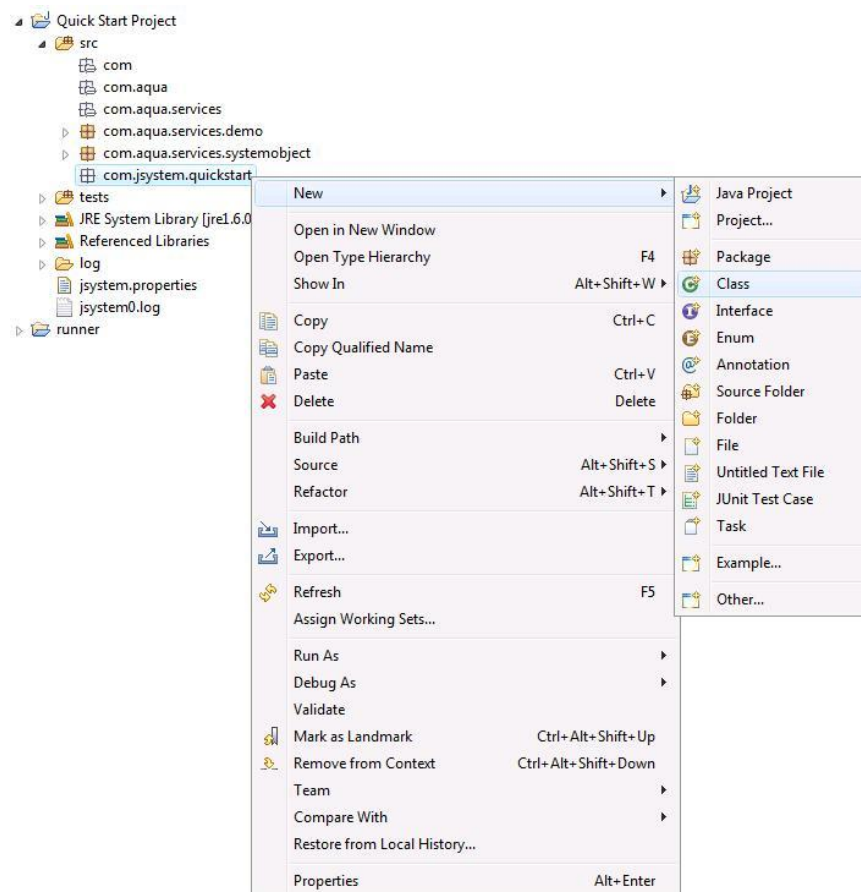


Figure 32: Creating a New Class

5. Name the class **"MyStation"**. The class should extend **"jsystem.framework.system.SystemObjectImpl"**

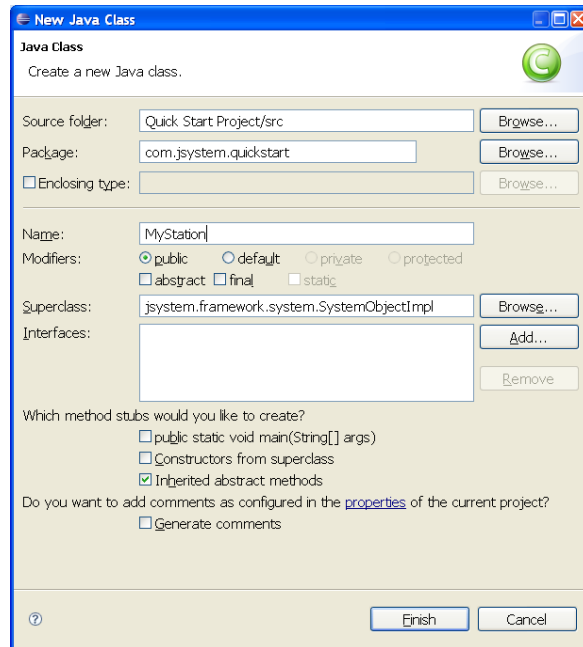


Figure 33: New Java Class Name

6. Now click the **"Finish"** button.
7. The first operation that the system object should implement is a ping; now add the following code example to the new Java class in the Eclipse project.

```
package org.jsystem.quickstart;
import jsystem.framework.system.SystemObjectImpl;

public class MyStation extends SystemObjectImpl {

    public void ping(String destination,int packetSize)
        throws Exception {

    }

}
```

Table 3: System Object Class Code Example

11.9.1.1 System Object Class Init and Close Methods

Two methods that you should get used to implementing when writing a system object are the **init()** method and the **close()** method. The **init()** method is called by the framework when the system object is initialized and the **close()** method is invoked when the system object is disposed

```
package org.jsystem.quickstart;
import jsystem.framework.system.SystemObjectImpl;

public class MyStation extends SystemObjectImpl {

    public void init() throws Exception {
        super.init();
        report.report("In init method");
    }

    public void close(){
        super.close();
        report.report("In close method");
    }

    public void ping(String destination,int packetSize)
        throws Exception {

    }

}
```

Figure 9: System Object with init() and close() methods

Note: When implementing the "init" and "close" methods, it is important to call the "init" and "close" of the super class, if this is not done correctly, the system object will not be initialized correctly.

In the example system object the usage of the reporter service is exemplified for the first time.

11.9.2 Extending System Objects Code Example

Eventually, our system object will interface a remote machine using the CLI driver.

The system object should know the remote machine IP, and CLI credentials (user name and password). Add these parameters to our system object:

```
package org.jsystem.quickstart;
import jsystem.framework.system.SystemObjectImpl;

public class MyStation extends SystemObjectImpl {
    private String host;
    private String userName;
    private String password;

    public void init() throws Exception {
        super.init();
        report.report("In init method");
    }

    public void close(){
        super.close();
        report.report("In close method");
    }

    public void ping(String destination,int packetSize)
        throws Exception {
    }

    public String getHost() {
        return host;
    }

    public void setHost(String host) {
        this.host = host;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Table 4: System object with cli related members

11.10 Incorporating a System Object in a Test

To be able to incorporate the **"MyStation"** system object in the test, we need to add it to the **"QuickSart"** project class path.

1. Right click on the "QuickStart" project and select the "Properties" menu item.

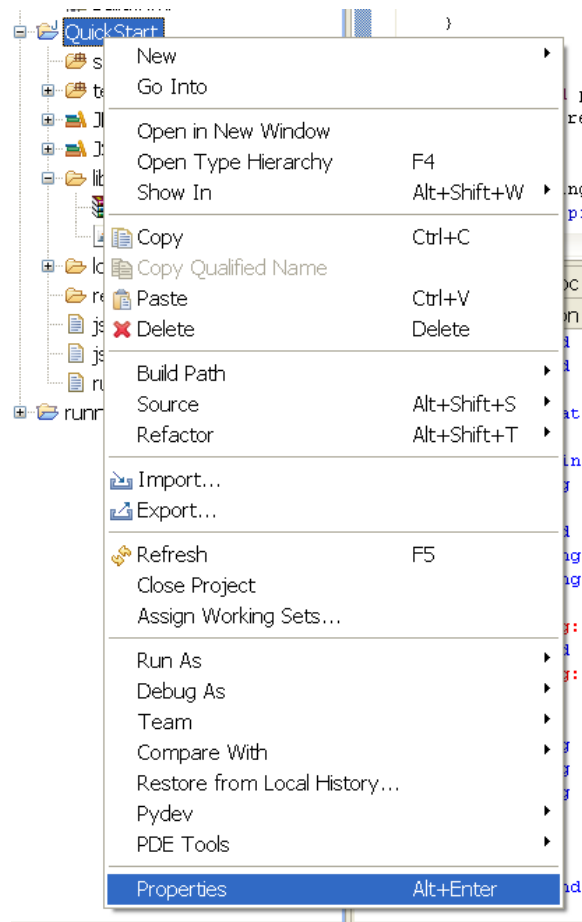


Figure 34: Project properties menu item

2. In the “Properties” tab select the “Java Build Path” item and in it, select the “Projects” tab

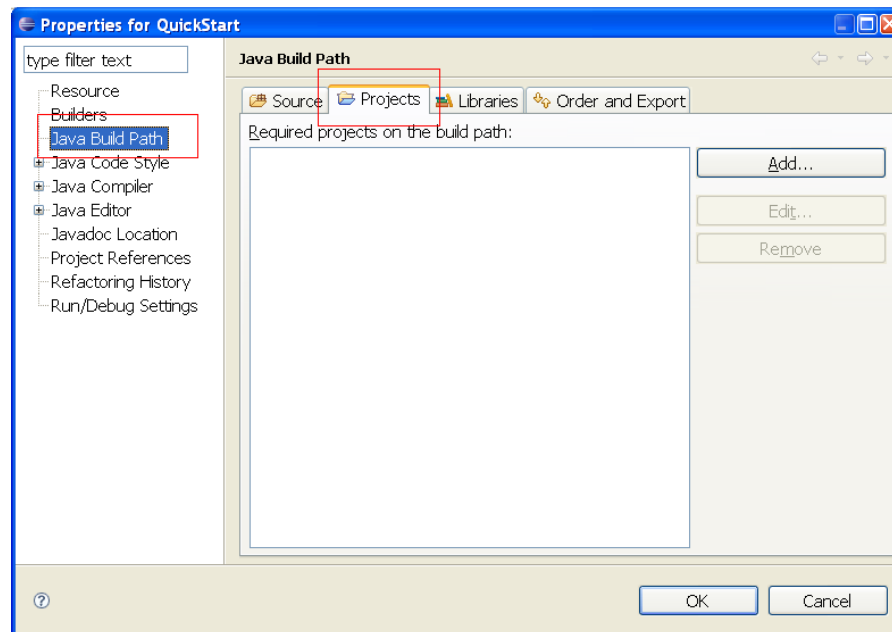


Figure 35: Project properties menu item

1. Press on the “Add” button and select the “MyStation” project

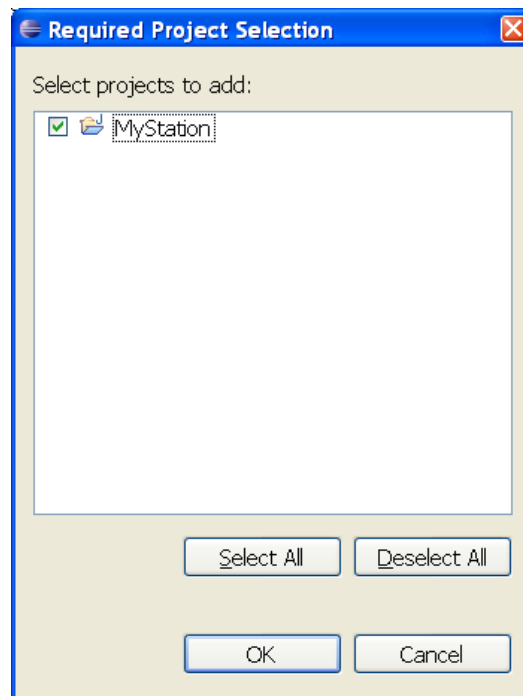


Figure 36: Required project selection

2. Press **"Ok"**. The **"MyStation"** project is not added to the list of projects in the classpath

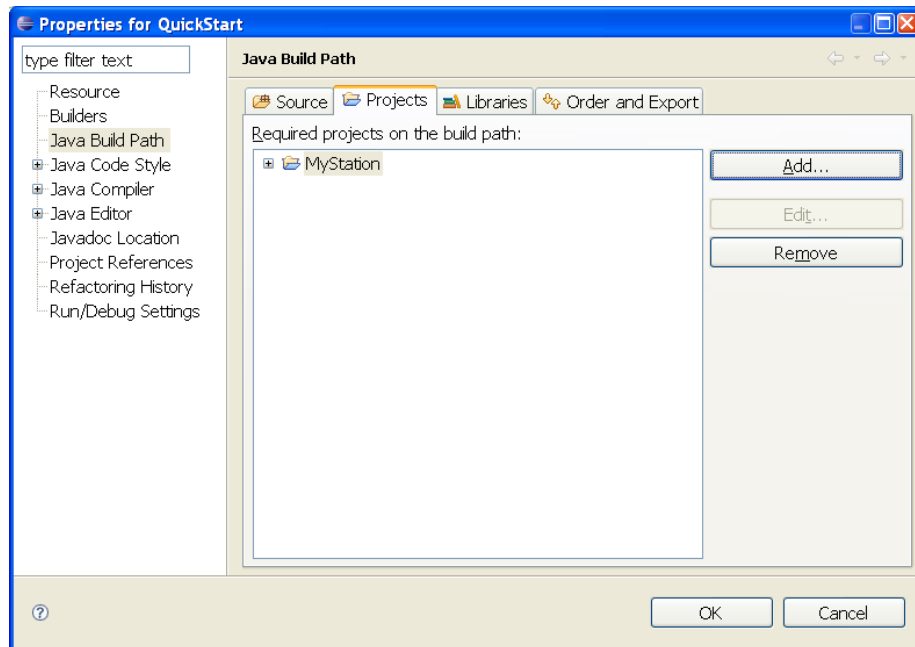


Figure 37: Java build tab after system object selection

3. Press **"Ok"** again.

In order to incorporate the system object in the test code, add a class member to the **"HostSanityTest"** test of type **"MyStation"**, initialize it in a **"before"** method by asking for an instance of the system object from the system and call the **ping** operation in the test method.

```

package org.jsystem.quickstart;
import org.junit.Before;
import org.junit.Test;
import org.jsystem.quickstart.MyStation;
import junit.framework.TestCase4;
public class HostSanityTest extends SystemTestCase4 {
    private String pingDestination;
    private int    packetSize;
    private MyStation myStation;
    @Before
    public void before() throws Exception{
        myStation =
            (MyStation)system.getSystemObject("my_station");
    }
    @Test
    public void pingFromDUT() throws Exception {
        report.report("Calling myStation ping operation");
        myStation.ping(getPingDestination(),getPacketSize());
    }

    public String getPingDestination() {
        return pingDestination;
    }
    public void setPingDestination(String pingDestination) {
        this.pingDestination = pingDestination;
    }
    public int getPacketSize() {
        return packetSize;
    }
    public void setPacketSize(int packetSize) {
        this.packetSize = packetSize;
    }
}

```

Table 5: Incorporating a System Object

At this stage initialization of the system object will fail, and thus, the test will fail. Let's make the test work and then I will tie things up and explain how it works

11.11 Adding SUT Configuration File

The missing link that prevents from the test to work is the SUT file.

The SUT file is a configuration file that defines the configuration of the system objects in the setup (in our case, DUT host name, cli user name and password)

Follow these steps to add the SUT file for our project:

1. Go to the **“Quick Start”** Project and **“Right Click”** on the SUT package icon in the package explorer window and select a new file from the menu.

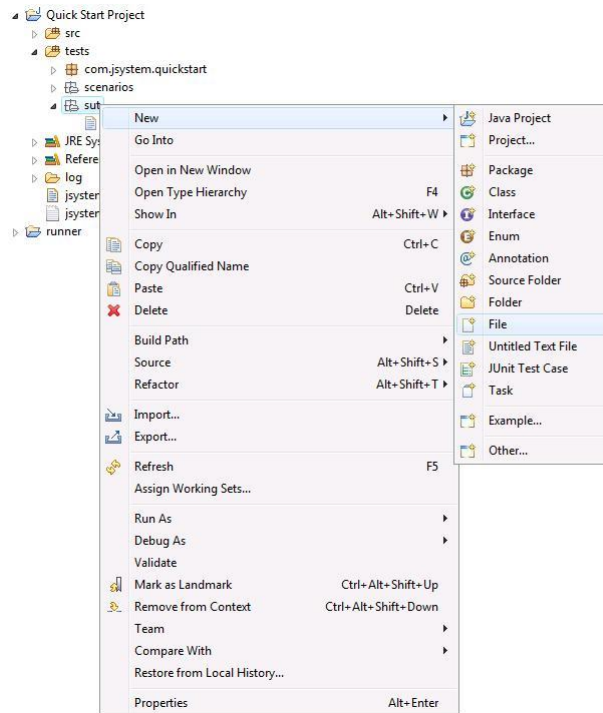


Figure 38: New SUT File Creation

Now set file name to be "setup.xml" and click the **"Finish"** button.

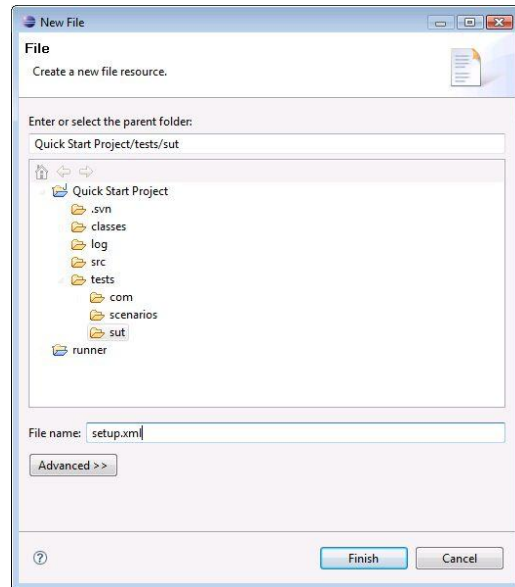


Figure 39: New SUT File Name

2. Now copy the following text into the file and save it

```
<sut>
  <my_station>
    <class>org.jsystem.quickstart.MyStation</class>
    <host>10.0.0.2</host>
    <userName>simpleuser</userName>
    <password>simpleuser</password>
  </my_station>
</sut>
```

Table 6: SUT Configuration File Code Example

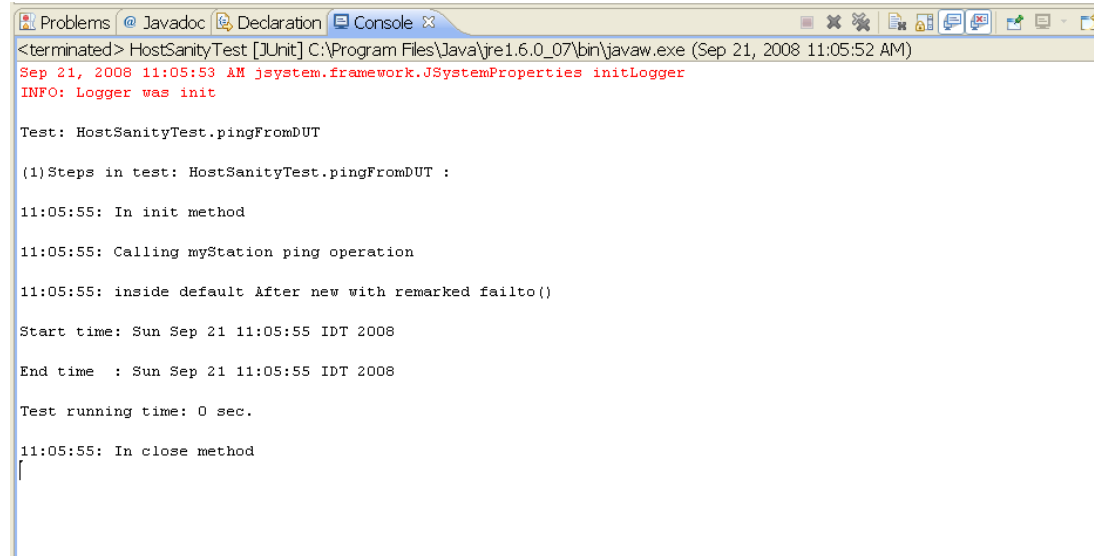
The **"host"** ip address, **"username"** and **"password"** are probably not applicable for your setup. We will leave them as is for now, since our system object doesn't use them at this stage.

11.11.1 Running a Test

Now that we have defined a SUT let's run the test and make sure it passes, after that we will understand how the magic is done.

In the package explorer view select the test class, "right click" on it and select Run/Debug As→JUnit Test.

The output of the execution should appear as it does in the Eclipse **"Console"** screen shot.



```
<terminated> HostSanityTest [JUnit] C:\Program Files\Java\jre1.6.0_07\bin\javaw.exe (Sep 21, 2008 11:05:52 AM)
Sep 21, 2008 11:05:53 AM jsystem.framework.JSystemProperties initLogger
INFO: Logger was init

Test: HostSanityTest.pingFromDUT

(1)Steps in test: HostSanityTest.pingFromDUT :

11:05:55: In init method

11:05:55: Calling myStation ping operation

11:05:55: inside default After new with remarked failto()

Start time: Sun Sep 21 11:05:55 IDT 2008

End time : Sun Sep 21 11:05:55 IDT 2008

Test running time: 0 sec.

11:05:55: In close method
|
```

Figure 40: Extensions Console Output Window

Make sure you see the log message **"In init method"**, it means that the system successfully activated the **"init"** method of the system object. Also make sure that the **"In close method"** is visible, indicating that the system object was closed on test termination.

Note: If you fail to initiate the system object, go to the trouble shooting section.

Instantiating the system object and managing its lifetime is part of the **"system"** service.

11.11.2 Understanding execution flow

The following section details the tests flow of execution.

1. Before test is executed, the **"before"** method is called. (This is the default JUnit behavior).
2. In the **"before"** method the code **"(MyStation)system.getSystemObject("my_station")"** is invoked, this triggers the following execution flow:
 - The system looks for a file called **"jsystem.properties"** in the current dir, if the file exists; Jsystem then searches for a property called the **"sutFile"**. The value of this property tells the system the name of the SUT file to use with test execution. Sut file is searched under the **"sut"** package.
 - In the event that the **"jsystem.properties"** file does not exist or the **"sutFile"** property does not exist in the properties file, the framework looks under the **"sut"** package directory, if there is only one SUT file under the package, Jsystem then automatically selects this file and sets the **"sutFile"** property, if there are several SUT files, Jsystem then opens a dialog that asks the user to select a SUT file. Once the user makes a selection JSystem updates the **"jsystem.properties"** file with the users selection.

- Now that the system has been synchronized with a SUT file, it now searches for the entity under the **<sut>** entity directory called **"my_station"**. This is the XML entity that defines the configuration of the system object instance identified by the name **"my_station"**.
- The system now searches under the system object entity for the **<class>** entity, it takes the value of this entity and using reflection it creates an instance of the class.
- The JSystem Automation Framework searches all other entities in the system object entity and looks for a corresponding setter method in the system object class.
For example, in the **<host>** entity, the system searches for a setter method called **"setHost"** in the class **"MyStation"**. If it finds the setter method, it activates it and passes to it the value of **"10.0.0.2"**. If a corresponding setter method is not found, the entity is ignored.
- At the end of the process the system object is instantiated and populated with values as defined in the SUT file.

11.11.3 Compiling and packaging the system object

When trying to refresh the JRunner and run the test we get an disturbing error indication:

Start: org.jssystem.quickstart.HostSanityTest.pingFromDUT	
Test: HostSanityTest.pingFromDUT	✓
(2)Steps in test: HostSanityTest.pingFromDUT :	✓
Params:	
PacketSize=0	✓
PingDestination=10.0.4.5	✓
12:35:35: Fail: org.jssystem.quickstart.MyStation	⚠
12:35:35: inside default After new with remarked failto()	⚠
Start time: Sun Sep 21 12:35:34 IDT 2008	✓
End time : Sun Sep 21 12:35:35 IDT 2008	✓
Test running time: 0 sec.	✓
End: org.jssystem.quickstart.HostSanityTest.pingFromDUT	⚠

Figure 41: Error when running test without compiling the system object

When we look at the full log we will se the following error:

```
java.lang.ClassNotFoundException: org.jssystem.quickstart.MyStation
    at java.net.URLClassLoader$1.run(URLClassLoader.java:200)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:251)
```

Figure 42: Exception when running test without compiling the system object

The runner doesn't know our system object.

Earlier in the tutorial we have added the system object project to the classpath of our tests project, this operation is relevant only for Eclipse. To add our system object to the runner's class path, we need to package it in a jar and distribute the jar to the **"lib"** folder

Compiling and packaging the system object in a jar is done by the **"build.xml"** ant file. Before activating the ant file some small changes should be done on it.

1. Open the build.xml file by double clicking on it.

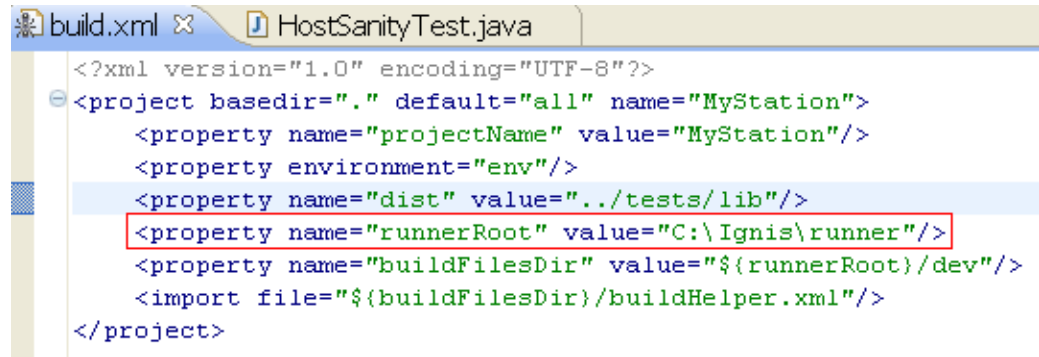


Figure 43:build.xml

2. Make sure that the **"runnerRoot"** property points to the runner root folder,
3. The **"dist"** property should points to the destination folder to where the system object jar should be distributed. Let's point the build to the lib folder of our tests project. The new value of the **"dist"** is: **"../QuickStart/lib"**. After making the change the build.xml should look like this:

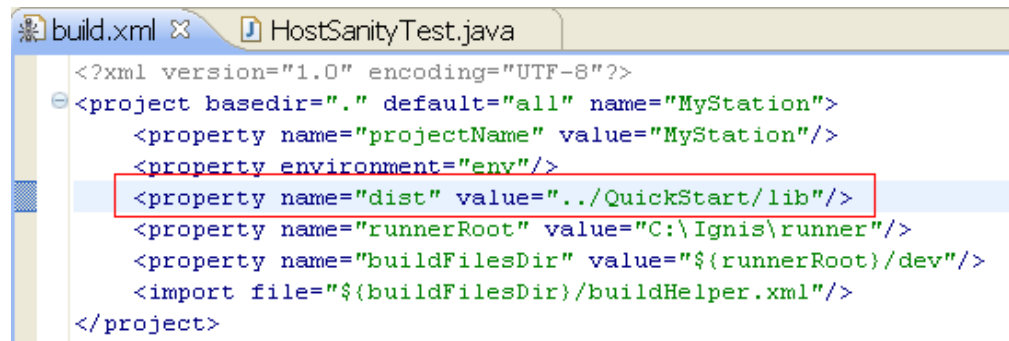
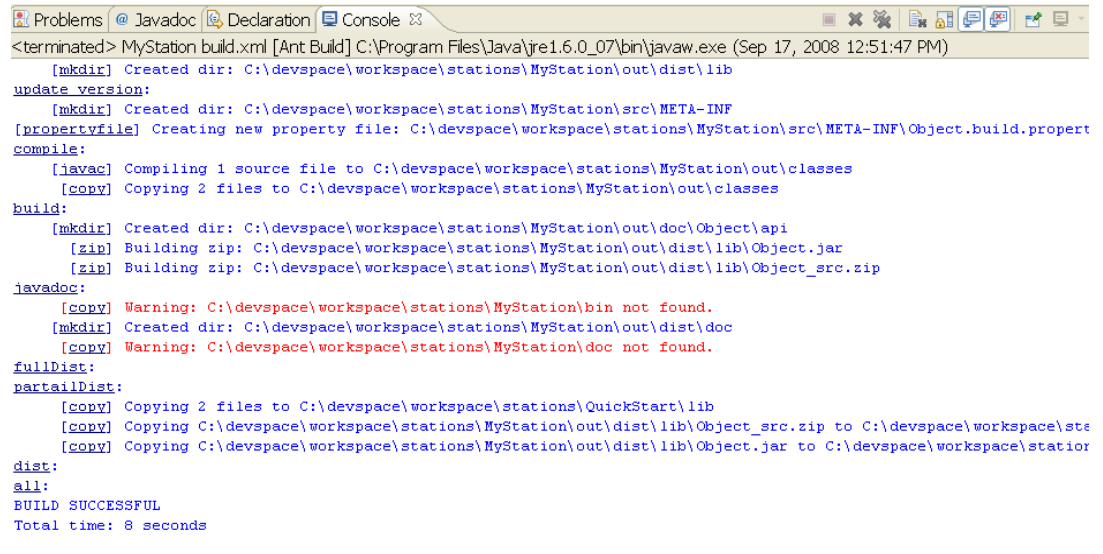


Figure 44:build.xml after pointing dist property to tests projects

4. Now right-click on the build.xml file and select **"Run As → Ant Build"**

5. The output of the build process should be:



```
<terminated> MyStation build.xml [Ant Build] C:\Program Files\Java\jre1.6.0_07\bin\javaw.exe (Sep 17, 2008 12:51:47 PM)
[mkdir] Created dir: C:\devspace\workspace\stations\MyStation\out\dist\lib
update_version:
[mkdir] Created dir: C:\devspace\workspace\stations\MyStation\src\META-INF
[propertyfile] Creating new property file: C:\devspace\workspace\stations\MyStation\src\META-INF\Object.build.properties
compile:
[javac] Compiling 1 source file to C:\devspace\workspace\stations\MyStation\out\classes
[copy] Copying 2 files to C:\devspace\workspace\stations\MyStation\out\classes
build:
[mkdir] Created dir: C:\devspace\workspace\stations\MyStation\out\doc\Object\api
[zip] Building zip: C:\devspace\workspace\stations\MyStation\out\dist\lib\Object.jar
[zip] Building zip: C:\devspace\workspace\stations\MyStation\out\dist\lib\Object_src.zip
javadoc:
[copy] Warning: C:\devspace\workspace\stations\MyStation\bin not found.
[mkdir] Created dir: C:\devspace\workspace\stations\MyStation\out\dist\doc
[copy] Warning: C:\devspace\workspace\stations\MyStation\doc not found.
fullDist:
partailDist:
[copy] Copying 2 files to C:\devspace\workspace\stations\QuickStart\lib
[copy] Copying C:\devspace\workspace\stations\MyStation\out\dist\lib\Object_src.zip to C:\devspace\workspace\stations\QuickStart\lib
[copy] Copying C:\devspace\workspace\stations\MyStation\out\dist\lib\Object.jar to C:\devspace\workspace\stations\QuickStart\lib
dist:
all:
BUILD SUCCESSFUL
Total time: 8 seconds
```

Figure 45:build output

Make sure the **"BUILD SUCCESSFUL"** message appears.

6. To make sure jar was created and copied to the **"Quick Start"** project, select the **"QuickStart"** project and press on F5 to refresh the view. Now open the lib folder and make sure the jar is there.

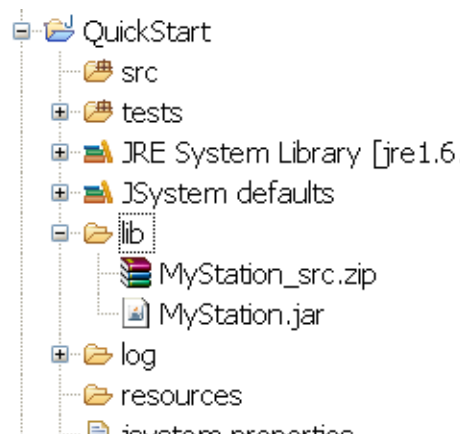


Figure 46:MyStation jar in lib folder of the tests project

Now refresh the runner and run the test again. This time the test should pass.

11.12 Step 9: Implementing the Ping Command

Now that the system object skeleton is ready, and it is incorporated into the test, the next step is to implement the “**ping**” operation.

In order to perform the implementation of the ping command, the “**CliConnection**” class that is part of the Cli package is used. Before writing the code and debugging it make sure to manage the work manually with the Cli agent (Telnet or SSH).

Reference: For a detailed explanation on the Cli agent.

11.12.1 Windows Code Example

Now add the following new code to the “**MyStation**” class in the project.

```
package org.jsystem.quickstart;
import com.aqua.sysobj.conn.CliCommand;
import com.aqua.sysobj.conn.CliConnectionImpl;
import com.aqua.sysobj.conn.WindowsDefaultCliConnection;
import jsystem.framework.system.SystemObjectImpl;

public class MyStation extends SystemObjectImpl {
    private String host;
    private String userName;
    private String password;
    private CliConnectionImpl connection;
    public void init() throws Exception {
        super.init();
        report.report("In init method");
        connection = new
            WindowsDefaultCliConnection(
                getHost(),getUserName(),getPassword());
        connection.init();
    }
    public void close(){
        super.close();
        report.report("In close method");
    }
    public void ping(String destination,int packetSize)
        throws Exception {
        CliCommand command =
            new CliCommand("ping " + destination + " -l " + packetSize);
        connection.handleCliCommand("Ping performed", command);
    }
    public String getHost() {
        return host;
    }
    public void setHost(String host) {
        this.host = host;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPassword() {
```

```
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Table 7: Windows Code Example

11.12.2 Revising Newly Added Code

1. Now that you have added a new class member to the system object of type **"CliConnectionImpl"** this member will take care of the CLI operation.
2. In the **"init"** method of the system object, we initialized the CliConnection entity.
3. Note that since the CliConnection entity is a system object, its initialization involves invoking the **"init"** method".
also, since the CliConnection is a system object it can be initialized from the SUT file as a nested system object. We will discuss it in the JSystem advanced issues tutorial.
4. In the ping method a new CliCommand object has been created with the ping command, the CliConnection then asks to execute the command.

11.12.3 Linux Code Example

```
01 package org.jssystem.quickstart;
02 import java.io.File;
03 import jssystem.framework.system.SystemObjectImpl;
04 import systemobject.terminal.Prompt;
05 import com.aqua.sysobj.conn.CliCommand;
06 import com.aqua.sysobj.conn.CliConnectionImpl;
07 import com.aqua.sysobj.conn.LinuxDefaultCliConnection;
08 public class MyStation extends SystemObjectImpl {
09     private String host;
10     private String userName;
11     private String password;
12     private CliConnectionImpl connection;
13     public void init() throws Exception {
14         super.init();
15         report.report("In init method");
16         connection = new LinuxDefaultCliConnection(getHost(),getUse
rName(),getPassword());
17         Prompt p = new Prompt("~$",false);
18         p.setCommandEnd(true);
19         connection.addPrompts(new Prompt[]{p});
20         connection.init();
21     }
22     public void close(){
23         super.close();
24         report.report("In close method");
25     }
26     public void ping(String destination,int packetSize) throws Ex
ception {
27         CliCommand command =
28             new CliCommand("ping " + destination +
" -s " + packetSize + " -c 3");
29         connection.handleCliCommand("Ping performed", command);
30     }
31     public void deployPatch(File patchPath) throws Exception {
32         report.report("In deploy patch operation");
33     }
34     public void updateHostsTable(String entryKey,String entryValu
e) throws Exception {
35         report.report("In update hosts file");
36     }
37     public String getHost() {
38         return host;
39     }
40     public void setHost(String host) {
41         this.host = host;
42     }
43     public String getUserName() {
44         return userName;
45     }
46     public void setUserName(String userName) {
47         this.userName = userName;
48     }
49     public String getPassword() {
50         return password;
51     }
```

```

52     public void setPassword(String password) {
53         this.password = password;
54     }
55 }

```

Table 8: Linux Code Example

11.12.4 The Difference between Linux and Windows Code

As can be seen in the two previous code examples above there are slight differences between the Linux and Windows code examples. The following example shows a section of Linux code with the difference marked in red.

```

15     report.report("In init method");
16     connection = new LinuxDefaultCliConnection(getHost(),getUserName(),getPassword());
17     Prompt p = new Prompt("~$",false);
18     p.setCommandEnd(true);
19     connection.addPrompts(new Prompt[]{p});
20     connection.init();
21 }
22 public void close(){
23     super.close();
24     report.report("In close method");
25 }
26 public void ping(String destination,int packetSize) throws Exception {
27     CliCommand command =
28         new CliCommand("ping " + destination + " -s " + packetSize + " -c 3");
29     connection.handleCliCommand("Ping performed", command);
30 }

```

Figure 10: Linux and Windows Differences

Behavioral Difference – The Linux ping command syntax and its default behavior is different from a Windows command, so the following differences are present.

The first change marked in red in lines 16 to 19, requires an explanation - Use the **"LinuxDefaultCliConnection"** command because the default shell prompt is different between various distributions of Linux, the machine prompt must be specifically defined and then added to the Cli connection. The second code line 28 is its syntax.

11.12.5 Running the Test from within Eclipse

Before running the test with the updated system object give the “**pingDestination**” and “**packetSize**” test parameters a default value as they appear in line 4 and 5 marked in red.

```
package org.jsystem.quickstart;
import org.junit.Before;
import org.junit.Test;
import org.jsystem.quickstart.MyStation;
import junit.framework.TestCase4;
public class HostSanityTest extends TestCase4 {
    private String pingDestination = "127.0.0.1";
    private int packetSize = 1024;
    private MyStation myStation;
    @Before
    public void before() throws Exception{
        myStation =
            (MyStation)system.getSystemObject("my_station");    }
    @Test
    public void pingFromDUT() throws Exception {
        report.report("Calling myStation ping operation");
        myStation.ping(getPingDestination(),getPacketSize());
    }

    public String getPingDestination() {
        return pingDestination;
    }
    public void setPingDestination(String pingDestination) {
        this.pingDestination = pingDestination;
    }
    public int getPacketSize() {
        return packetSize;
    }
    public void setPacketSize(int packetSize) {
        this.packetSize = packetSize;
    }
}
```

Table 9: Parameters Default Value Code Example

11.12.6 Debugging the Test

Now, “**right click**” on the test class and select Debug As→JUnit Test Case.

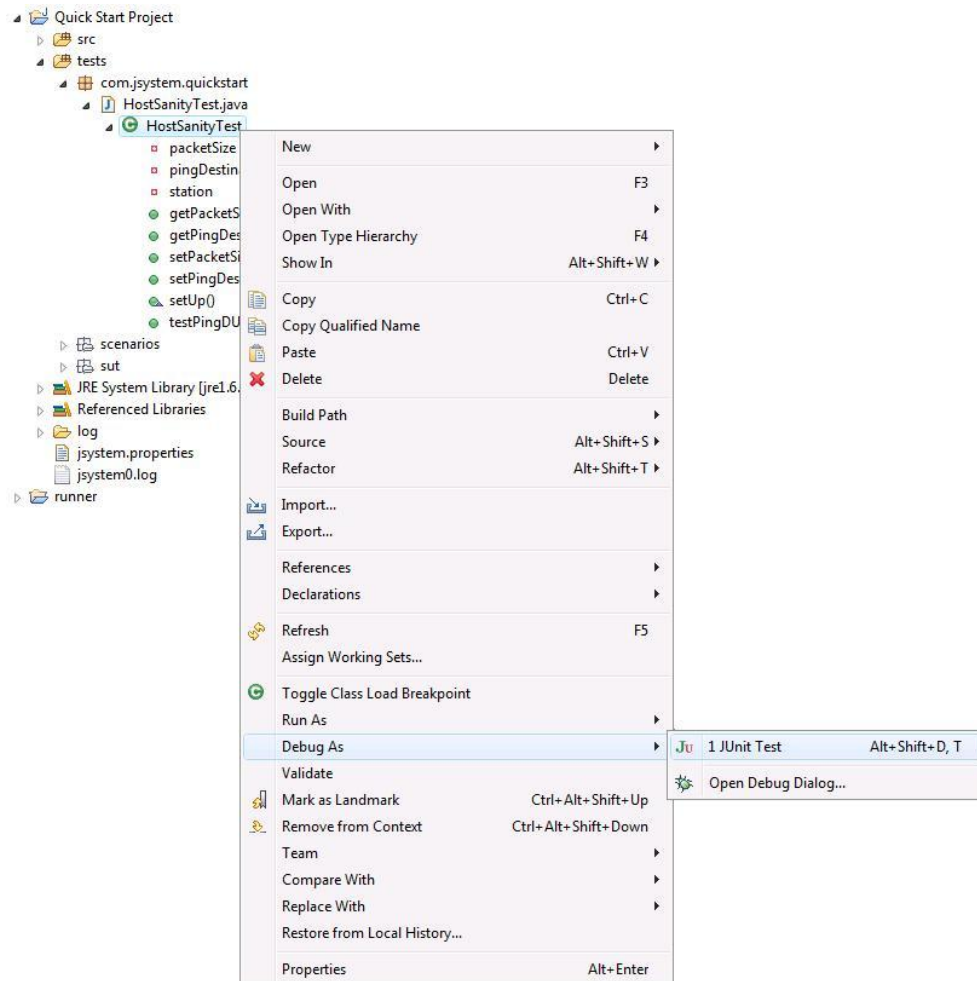


Figure 47: Debugging Test Menu

11.12.6.1 Eclipse Console Output

Below is test execution output, let's review it

```
01 Test: HostSanityTest.pingFromDUT
02 (1)Steps in test: HostSanityTest.pingFromDUT :
03 14:11:16: In init method
04 14:11:16: Init cli, host: 10.0.0.6
05 Welcome to Microsoft Telnet Service
06 login: simpleuser
07 password:
08 *=====
09 Welcome to Microsoft Telnet Server.
10 *=====
11 C:\Documents and Settings\simpleuser>
12 14:11:18: Calling myStation ping operation
13 ping 127.0.0.1 -l 1024
14
15 Pinging 127.0.0.1 with 1024 bytes of data:
16
17 Reply from 127.0.0.1: bytes=1024 time<1ms TTL=128
18 Reply from 127.0.0.1: bytes=1024 time<1ms TTL=128
19 Reply from 127.0.0.1: bytes=1024 time<1ms TTL=128
20 Reply from 127.0.0.1: bytes=1024 time<1ms TTL=128
21
22 Ping statistics for 127.0.0.1:
23     Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
24     Approximate round trip times in milli-seconds:
25         Minimum = 0ms, Maximum = 0ms, Average = 0ms
26 C:\Documents and Settings\simpleuser>
27 14:11:22: Ping performed
28 14:11:22: inside default After new with remarked failto()
29 Start time: Sun Sep 21 14:11:16 IDT 2008
30 End time   : Sun Sep 21 14:11:22 IDT 2008
31 Test running time: 5 sec.
32 14:11:22: In close method
```

Table 10: Telnet Console output

1. Lines 1-2 jsystem initiation messages
2. Lines 3-11 – MyStation system object initialization. System object initialization includes CLICConnection connection initialization.
3. Lines 12-27 test execution (which includes activation of the system object)
4. Lines 28-31 test tear down
5. Line 32: after teardown, close method of the system object is closed.

11.12.7 Running the test from the JRunner

Once a successful test run has been performed from within the Eclipse environment open the JRunner, refresh the project, and run the scenario again.

Note the following

1. The test progress and Cli operations are shown in the reporter tab.
3. A full Cli dump can be seen in JRunner console.
4. Open the HTML log file and browse to the test log and go to the Cli command and command output.
5. Using the JRunner try to change test parameter values and observe how the test behaves.

11.13 Step 10: Performing Analysis

Once the **"ping"** command has been activated it must be checked in order to ascertain if the test has failed or passed, this is achieved by observing the output of the command.

The next step is to implement a simple analysis of the ping operation. To make it simple we will assume that if the text **"(0% loss)"** is retrieved as an output of the operation, it means the ping succeeded.

Like previously done in this tutorial, first we will add the analysis code to the system object and the test, and then I will explain JSystem analysis principals.

11.13.1 System Object code

Add the **"setTestAgainstObject"** method to the **"MyStation"** system object **"ping"** command.

```
public void ping(String destination,int packetSize) throws
    Exception {
    CliCommand command = new CliCommand("ping " + destination +
                                         " -l " + packetSize);

    connection.handleCliCommand("Ping performed", command);
    setTestAgainstObject(command.getResult());
}
```

Table 10: System Object Ping Operational with analysis support

11.13.2 Test Code

Next invoke the “**analyze**” method with the “**FindText**” analyzer.

```
package org.jsystem.quickstart;
import jsystem.extensions.analyzers.text.FindText;
import junit.framework.TestCase4;

import org.junit.Before;
import org.junit.Test;
public class HostSanityTest extends SystemTestCase4 {
    private String pingDestination = "127.0.0.1";
    private int    packetSize = 1024;
    private MyStation myStation;

    @Before
    public void before() throws Exception{
        myStation =
            (MyStation)system.getSystemObject("my_station");
    }

    @Test
    public void pingFromDUT() throws Exception {
        report.report("Calling myStation ping operation");
        myStation.ping(getPingDestination(),getPacketSize());
        myStation.analyze(new FindText("{0% loss}"));
    }

    public String getPingDestination() {
        return pingDestination;
    }
    public void setPingDestination(String pingDestination) {
        this.pingDestination = pingDestination;
    }
    public int getPacketSize() {
        return packetSize;
    }
    public void setPacketSize(int packetSize) {
        this.packetSize = packetSize;
    }
}
```

Table 11: Test Ping DUT Code Example

11.13.3 Analysis principals and flow

Analyzers provide an easy way to verify system object operations. Once operation is performed, the result of the operation is defined to be the object to be analyzed using the “**setTestAgainstObject**” method. Later in the flow, the test invokes an “**Analyzer**” on the **testAgainstObject** by activating the “**analyze**” method and supplying it the requested analyzer.

The analysis service comprises an architecture that supports two kinds of users, an advanced programmer that writes the system object and analyzers, and users that use the analyzers to examine operation results.

In our case, the results of the ping command are provided to the **"testAgainstObject"** entity by activating the **"setTestAgainstObject"** method in the system object. The **"analyze"** method of the system object is then activated from within the test method. The analyze method is inherited from the **"SystemObjectImpl"** base class.

11.13.4 Play the Scenario

We have made changes in the system object class so we need to activate the build.xml. Once the Mystation jar built and deployed, open the JRunner, refresh it, and change the **"pingDestination"** parameter value to a destination that doesn't exist for example (**"dummy_dest123.com"**).

After executing the test the JRunner will display the following error.

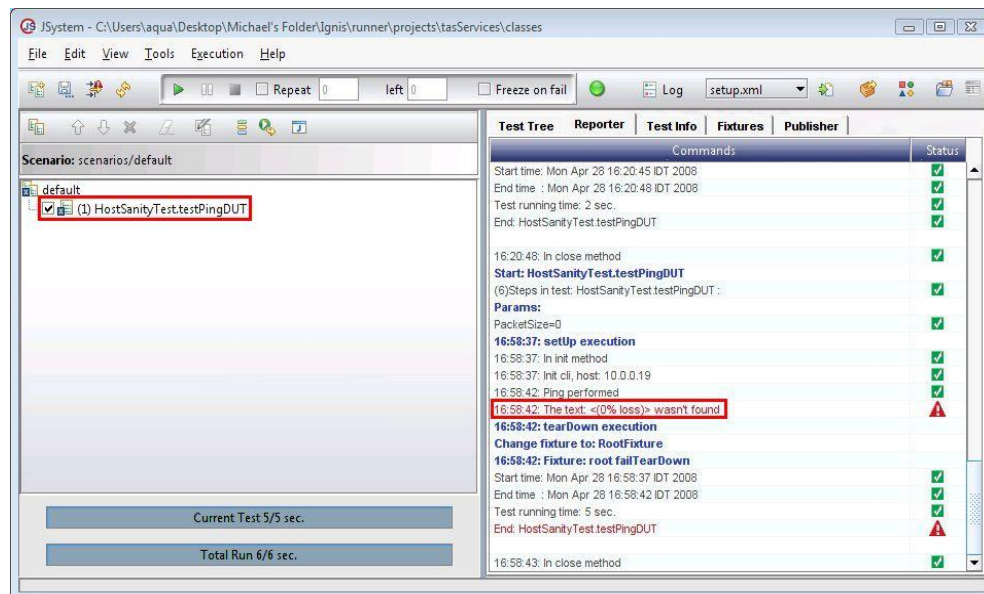



Figure 48: Scenario Execution

Note: The test has failed and in the **"Reporter"** tab we see the log message: **"the text <(0% loss)> wasn't found"**.

Now press the  “**View Log**” button in order to generate an HTML report of the test that has been run in the JRunner.

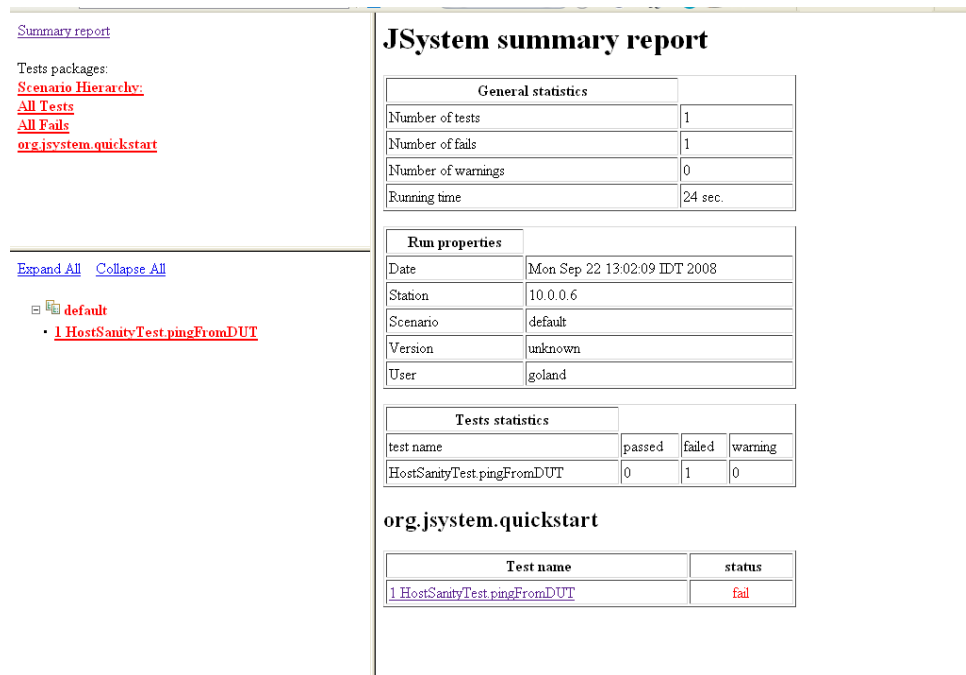


Figure 49: Scenario Execution Log

In the test log the message appears as it was shown in the JRunner reporter tab; now drill down into this message by pressing on it and you will see the exact output of the ping command appears and the reason for the test failure.

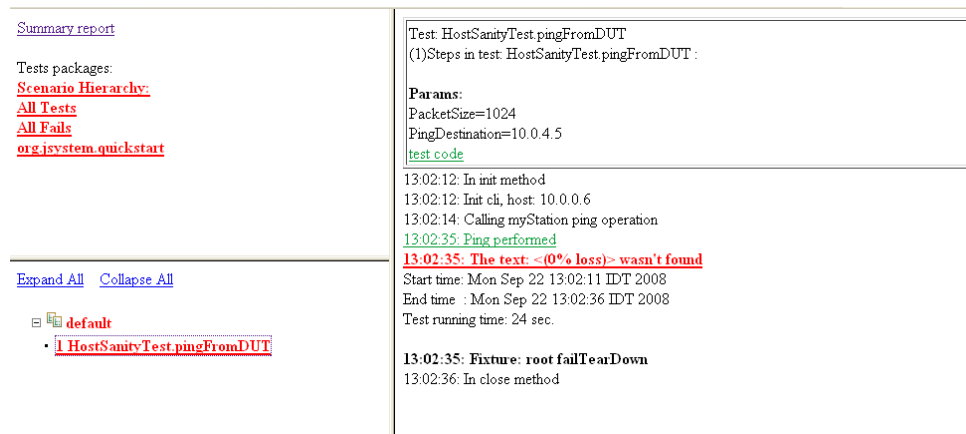


Figure 50: Detailed Scenario Execution Log

Now change the “**pingDestination**” back to an IP that the DUT can reach - 127.0.0.1 for example, replay the scenario. After performing the execution the test will appear without errors.

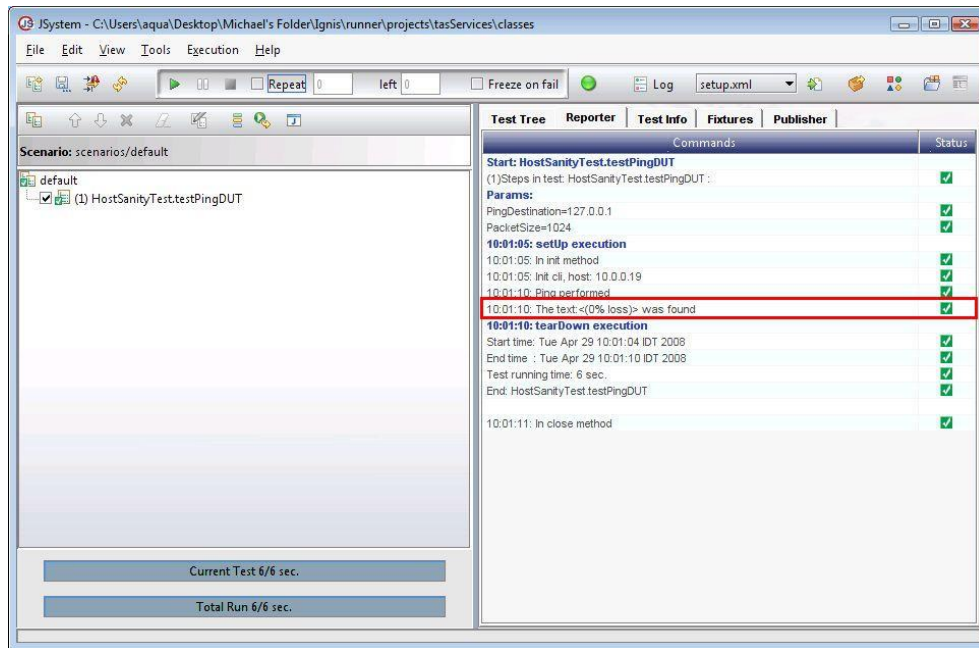


Figure 50: JSystem Successful Test Execution

The test passed and in the reporter tab we see the test **"the text <(0% loss)> was found"**

Note: *Want to clean the reporter tab and log file between runs? Select Tools → Init Reporters.*

11.14 Step 11: Writing the Fixtures

The last stage of the “**Quick Start Project**” will incorporate fixtures.

A fixture is a special type of class that implements code that configures the setup and restores the setup to its initial state (as opposed to testing code which is implemented by the test) the purpose of fixtures is to help the author of the automation project to separate between code that configures the setup and code that tests the setup and to optimize the navigation of the system between configuration states.

After writing fixtures the automation project author assigns a fixture to a test/scenario. Before the system executes the test/scenario, it checks the fixture association and then it navigates to this fixture. When navigating to the fixture the configuration code is executed.

In the example project you will write two fixtures: one that installs a patch on the DUT and another that configures the host's file of the DUT.

All operations that are done on the DUT should be implemented in the system object so we will start by adding the skeleton of these operations to the system object, then we will write the fixture that invokes these operations, finally you will learn how to associate the fixture with the test.

Note: Because of the complexity involved in implementing the code that installs a patch and updates the hosts file on our DUT (and since this is not the purpose of this tutorial), we will construct the skeleton without adding more drivers.

11.14.1 Adding Operations to the System Objects

Now add the “**deployPatch**” and the “**updateHostsTable**” operations skeleton code to the “**MyStations**” system object.

11.14.1.1 System Object code

```
package org.jsystem.quickstart;
import java.io.File;
import jsystem.framework.system.SystemObjectImpl;
import com.aqua.sysobj.conn.CliCommands;
import com.aqua.sysobj.conn.WindowsDefaultCliConnection;
public class MyStation extends SystemObjectImpl {
    private String host;
    private String userName;
    private String password;
    private WindowsDefaultCliConnection connection;
    public void init() throws Exception {
        super.init();
        report.report("In init method");
        connection = new
            WindowsDefaultCliConnection(getHost(),
                                         getUserName(),
                                         getPassword());

        connection.init();
    }
    public void close(){
        super.close();
        report.report("In close method");
    }
    public void ping(String destination,int packetSize)
        throws Exception {
        CliCommand command = new CliCommand("ping " +
                                             destination + " -l " + packetSize);
        connection.handleCliCommand("Ping performed", command);
        setTestAgainstObject(command.getResult());
    }
    public void deployPatch(File patchPath) throws Exception {
        report.report("In deploy patch operation");
    }
    public void updateHostsTable(String entryKey,String entryValue)
        throws Exception {
        report.report("In update hosts file");
    }
    public String getHost() {
        return host;
    }
    public void setHost(String host) {
        this.host = host;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPassword() {
        return password;
    }
}
```


Table 12: Adding Operations Code Example

11.14.2 Implementation of a Fixture

In order to implement a fixture the following steps must be taken.

1. The fixture class should extend the "**jssystem.framework.fixture.Fixture**" class.
2. The fixture class should implement the method "**setUp()**" and optionally "**tearDown()**" and "**failTearDown()**". The "**setUp**" method is invoked when navigating to the fixture, "**tearDown**" method it is invoked when navigating from the fixture to another fixture. The "**failTearDown**" method is invoked when navigating to another fixture in a case of test failure.
3. Fixtures like tests are part of the tests project hence, in our case, our first fixture should be added to the "**QuickStart**" project.

Now go to the tests folder of the "**QuickStart**" project and add new class by "**right clicking**" on the "**org.jssystem.quickstart**" package, name it "**InstallPatch**", the class should extend "**jssystem.framework.fixture.Fixture**" and click "**Finish**".

Copy the following code into the newly created class folder in your project and run and update.

```
package org.jssystem.quickstart;
import java.io.File;
import jssystem.framework.fixture.Fixture;
public class InstallPatch extends Fixture {
    private MyStation station;
    public InstallPatch(){
    }
    public void setUp() throws Exception {
        station = (MyStation)system.getSystemObject("my_station");
        station.deployPatch(new File("Path to patch file"));
    }
    public void tearDown() throws Exception {
    }
    public void failTearDown() throws Exception {
    }
}
```

Table 13: Fixture Code Example

11.14.3 Fixture execution

To see our new fixture in action, first we have to re-build the system object by running the build.xml. Now associate the scenario with the newly created fixture. Go to the JRunner and press on the 🔄 “refresh” button. In the tests tree you will now see the newly created fixture:

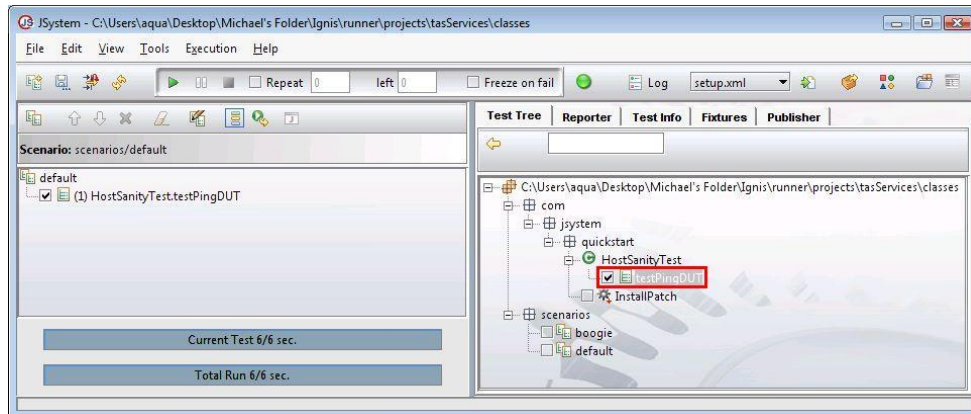


Figure 51: Initial Patch JSystem Example

Select the fixture and add it to the scenario.

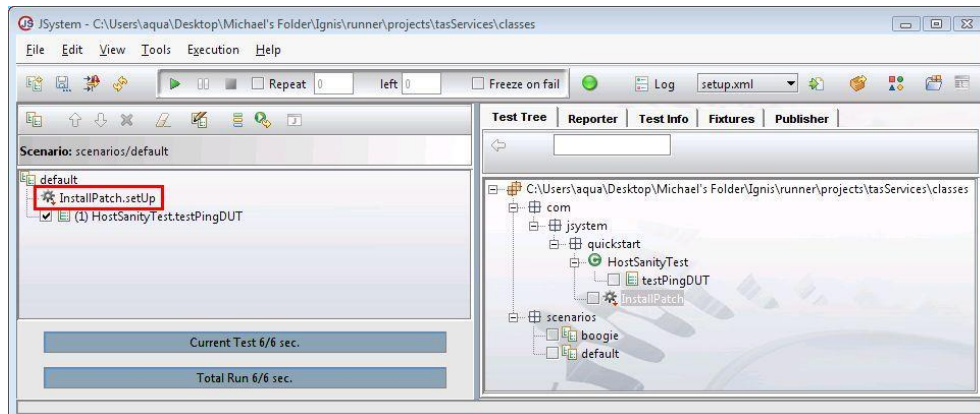


Figure 52: Initial Patch Setup Example

Now play the scenario and look at the log file.

14:33:33: Fixture: InstallPatch setUp

14:33:34: In init method

14:33:34: Init cli, host: 127.0.0.1

14:33:37: **In deploy patch operation**

14:33:37: setUp execution

14:33:40: Ping performed

14:33:40: The text:<(0% loss)> was found

14:33:40: tearDown execution

Start time: Wed Apr 16 14:33:33 IDT 2008

End time : Wed Apr 16 14:33:40 IDT 2008

Test running time: 7 sec.

14:33:40: In close method

Table 14: Run Fixture Example

Notice that the first thing that JRunner does is to navigate to the fixture and executes the "**setUp**" method, the JRunner then executes the test.

11.14.4 Associating a test with a Fixture

When associating a scenario with a fixture the fixture is associated with all the tests in the scenario. Another way to associate a test with a fixture is programmatically, this done by calling the **"setFixture"** method in the test constructor.

```
1 public HostSanityTest () {  
2     setFixture(InstallPatch.class);  
3 }
```

Table 15: Associating a test with a Fixture Code Example

Open the JRunner and remove the fixture from the scenario by selecting the fixture and pressing the **✖ "delete"** button.

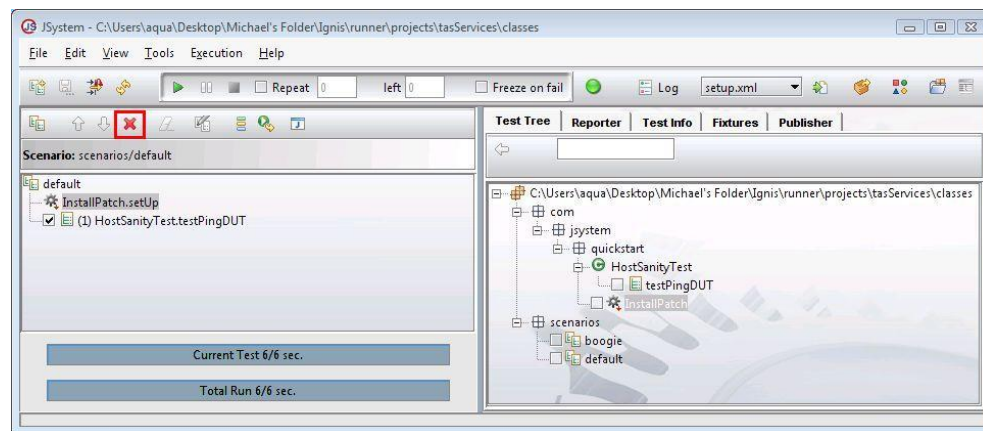


Figure 52: Remove Button in the JRunner

Now **🔄 "Refresh"** the JRunner and **▶ "Play"** the scenario. A brief look at the log file and the **"Reporter"** tab will show you that the fixture was executed.

11.14.4.1 Fixture Hierarchy Structures

In order to start understanding the power of fixtures we will incorporate another fixture in our project,

We will create a new fixture, “**ConfigureHostsTable**”, set the parent of this fixture to be the “**InstallPatch**” fixture. In order to set the parent of the fixture, invoke the “**setParentFixture**” method in the fixture’s constructor.

```
package org.jsystem.quickstart;
import jsystem.framework.fixture.Fixture;
public class ConfigureHostsTable extends Fixture {
    private MyStation station;
    public ConfigureHostsTable() {
        setParentFixture(InstallPatch.class);
    }
    public void setUp() throws Exception {
        station = (MyStation)system.getSystemObject("my_station");
        station.updateHostsTable("anotherLocalHost","127.0.0.1");
    }
    public void tearDown() throws Exception {
    }
}
```

Table 16: Fixture Hierarchy Code Example

Now go to the JRunner and 💰 “**Refresh**” it, go to the fixtures tab, there you can see fixtures runtime hierarchy



Figure 53: Fixtures hierarchy

Next, associate the “**default**” scenario with the “**ConfigureHostsTable**” fixture and play the scenario.

The log file output appears as follows.

15:08:45: Fixture: InstallPatch setUp

15:08:46: In init method
15:08:46: Init cli, host: 127.0.0.1
15:08:48: In deploy patch operation

15:08:48: Fixture: ConfigureHostsTable setUp

15:08:48: In update hosts file

15:08:48: setUp execution

[15:08:51: Ping performed](#)
[15:08:51: The text:<\(0% loss\)> was found](#)

15:08:51: tearDown execution

Start time: Wed Apr 16 15:08:45 IDT 2008
End time : Wed Apr 16 15:08:51 IDT 2008
Test running time: 6 sec.
15:08:51: In close method

Table 17: Configuration Table

Before the JRunner executes the test it navigates to the **"ConfigureHostsTable"**, the navigation involves executing the **"setUp"** method of the **"InstallPatch"** fixture and then the **"setUp"** method of the **"ConfigureHostsTable"** fixture.

11.14.5 More about fixtures

Benefiting from JSystem fixtures capabilities requires experience with JSystem. The **"JSystem Services"** chapter describes in details the fixtures module, before reading this chapter is recommended to work for a while with JSystem.

11.15 Quick start project summary

Let's sum-up the work that we have done in the QuickStart project

1. We have started with setting up the workspace and creating the system object project and tests project
2. We have continued with writing the skeleton of the test and then added to the test user parameters
3. We have written system object skeleton and incorporated the skeleton in the test
4. Incorporated CLI in the ping command.
5. Added analysis to the ping command results
6. Added fixtures to our projects.

11.15.1 What's next

The quick start project is a preamble to JSystem capabilities, but it is enough to start writing your automation project. If you feel you need to know more about JSystem read the **"JSystem Services"** chapter and then the **"JRunner Functionality"** chapter, otherwise start writing your tests and go back to the book when required.