Borland®

THE OPEN ALM COMPANY

SilkTest® 2008 R2

Silk4J Advanced User Guide

Borland Software Corporation
8310 North Capital of Texas Hwy
Building 2, Suite 100
Austin, Texas 78731
http://www.borland.com

Contents

Silk4J

Silk4J enables you to create functional tests using the Java programming language. Silk4J provides a Java runtime library that includes test classes for all the classes that Silk4J supports for testing. This runtime library is compatible with JUnit, which means you can leverage the JUnit infrastructure and run Silk4J tests. You can also use all available Java libraries in your testcases.

The testing environments that Silk4J supports include:

- Adobe Flex applications
- Java SWT applications
- Windows Presentation Foundation (WPF) applications
- Windows API-based client/server applications
- xBrowser applications

Sample Projects and Scripts

Use the sample projects and scripts that Silk4J provides to view typical script configurations and testcase execution.

Import the sample project into your Eclipse workspace and review the scripts. After you review the samples, get started creating your testcases. You can modify these throughout the testing cycle as necessary.

Note: The Eclipse integrated development environment (IDE) project includes the script that is used to demonstrate the tasks in the *Getting Started Tutorial*.

Environment	File Location
Adobe Flex	`<SilkTest_install_directory>\ng\samples\flex\Flex Store (Silk4J)`
Java SWT	`<SilkTest_install_directory>\ng\samples\java\swt\SWT Test Application (Silk4J)`
Windows API-based	`<SilkTest_install_directory>\ng\samples\win32\Calc (Silk4J)`
Windows Presentation Foundation (WPF)	`<SilkTest_install_directory>\ng\samples\dotnet\WPF Calc (Silk4J)`
xBrowser	`<SilkTest_install_directory>\ng\samples\xBrowser\GMO (Silk4J)`
Eclipse integrated development environment (IDE)	`<SilkTest_install_directory>\ng\samples\java\swt\Eclipse (Silk4J)`

Note: The sample scripts contain a path in the basestate to the location of the application under test. If SilkTest is not installed in the default location, adapt the paths in the test files to reference the correct location.

Related Topics

- *Getting Started Tutorial* on page 6

Getting Started Tutorial

This tutorial provides a step-by-step introduction to Silk4J.

👉 **Important:** To successfully complete this tutorial you will need basic knowledge of Java, JUnit, and the Eclipse IDE.

For the sake of simplicity, this guide assumes that you are using the Eclipse IDE with the Silk4J feature installed.

For additional information about Silk4J, refer to the *Silk4J User Guide*. To view the guide, in Eclipse choose **Help ➤ Help Contents** and then select Silk4J User Guide.

Related Topics

- *Starting the SilkTest Open Agent* on page 6
- *Creating a Silk4J Project* on page 6
- *Creating a Test Case for the Getting Started Tutorial* on page 7
- *Running Test Cases* on page 15

Starting the SilkTest Open Agent

Before you can create a test case or run a sample script, you must start the SilkTest Open Agent.

Choose **Start ➤ Programs ➤ Borland ➤ SilkTest <version> ➤ SilkTest Open Agent**.
The SilkTest Open Agent icon displays in the system tray.

Related Topics

- *Getting Started Tutorial* on page 6

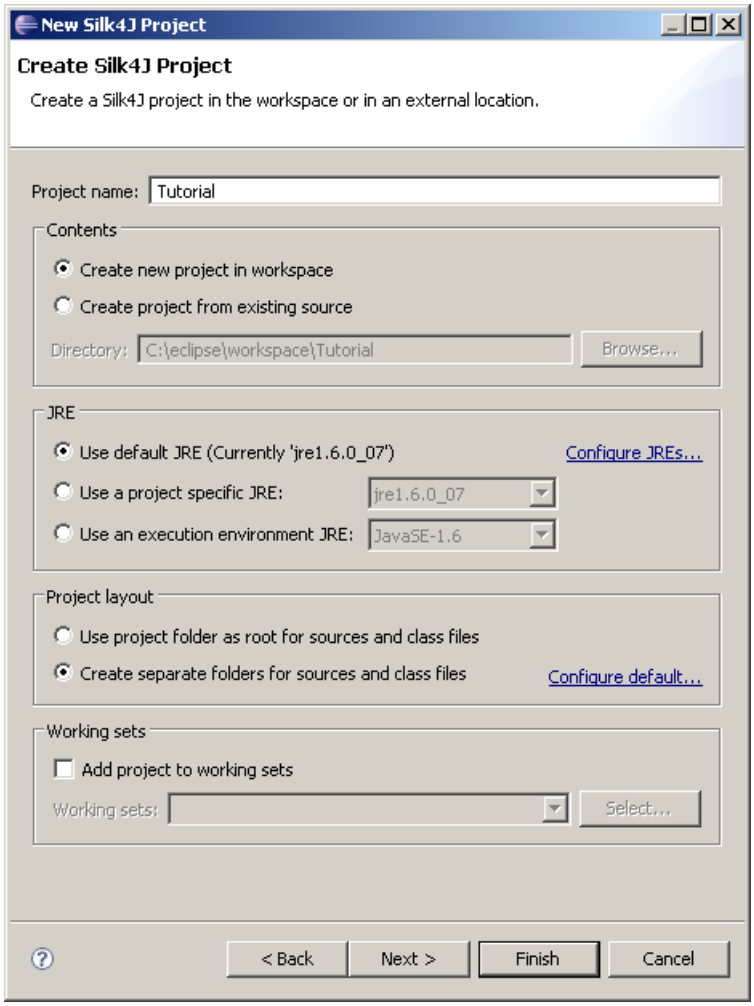
Creating a Silk4J Project

When you create a Silk4J project using the **Create Silk4J Project** wizard, the wizard contains the same options that are available when you create a Java project using the **New Java Project** wizard. Additionally, the Silk4J wizard automatically makes the Java project a Silk4J project.

For additional information about settings within the wizard, press F1 in the wizard.

1. In the Eclipse workspace, choose **File ➤ New ➤ Project.**
   The **New Project** wizard opens.
2. Expand the **Silk4J** folder and select **Silk4J Project**.
3. Click **Next**.
   The **New Silk4J Project** page opens.
4. In the **Project name** text box, type a name for your project.

5. Accept the default settings for the remaining options.



6. Click **Next** and specify any other settings that you require.
   Press F1 to access help for the settings on this page, if necessary.
7. Click **Finish**.
   A new Silk4J project is created that includes the JRE system library and the required `.jar` files, `silktest-jtf-nodeps.jar` and the `junit.jar`.

Related Topics

- *Getting Started Tutorial* on page 6

Creating a Test Case for the Getting Started Tutorial

Creating a test case involves several steps. The following tasks use an example to walk you through how to test an Eclipse integrated development environment (IDE). At the end of each task, a sample script shows the code that corresponds with the steps that were illustrated. The sample script builds with each task and the final task, *Creating a Test Method*, shows the entire script from beginning to end.

For a streamlined version of how to create a test case with less sample code, see *Creating Test Cases* in the *Managing Test Cases* section of the Silk4J User Guide.

Related Topics

- *Getting Started Tutorial* on page 6
- *Creating a JUnit Test Case for the Getting Started Tutorial* on page 8
- *Adding a Base State for a Test Case for the Getting Started Tutorial* on page 8
- *Creating a Test Method for the Getting Started Tutorial* on page 10

Creating a JUnit Test Case for the Getting Started Tutorial

1. Choose **File ➤ New ➤ JUnit Test Case**.
   The **New JUnit Test Case** dialog box opens.
2. Ensure the **New JUnit 4 test** option is selected.
   This option is selected by default.
3. In the **Package** text box, specify the package name.

   By default, this text box lists the most recently used package. If you do not want to use the default package, choose one of the following:

   - If you have not created the package yet, type the package name into the text box.
   - If you have created the package already, click **Browse** to navigate to the package location and then select it.
4. In the **Name** text box, specify the name for the test case.
5. Click **Finish**.
   The new class file opens with code similar to the following:

   ```
   package com.borland.demo;

   public class DynamicObjectRecognitionDemo {

   }
   ```

   where `com.borland.demo` is the package that you specified and `DynamicObjectRecognitionDemo` is the class that you specified.

Related Topics

- *Creating a Test Case for the Getting Started Tutorial* on page 7

Adding a Base State for a Test Case for the Getting Started Tutorial

The base state makes sure that the application that you want to test is running and in the foreground. This ensures that tests will always start with the same application state, which makes them more reliable. In order to use the base state, it is necessary to specify what the main window looks like and how to launch the application that you want to test if it is not running. Creating a base state is optional. However, it is recommended as a best practice.

This task illustrates how to create a base state for the Eclipse IDE. For a detailed version of how to add a base state for each technology type, see *Adding a Base State* in the *Managing Test Cases* section of the Silk4J User Guide.

1. Open the file in which you want to include the base state.

   👉 **Note:** The Java SWT sample application provided with Silk4J includes a generic helper class, `BaseStates.java`. If you are testing a Java SWT application, you might want to modify the sample file and use this helper class.

2. Create a member instance named `desktop`, which acts as the central entry point for your test.
   For example, type:
   ```
   private final Desktop desktop = new Desktop();
   ```

3. Press Ctrl+Shift+O.
   Eclipse automatically adds and updates all the required import packages to include.

   👉 **Important:** Press Ctrl+Shift+O every time you add a line of code to ensure that the required import packages are included in the script.

4. Create a member instance called `Shell` for storing the reference to the main window of the application that you want to test.

   Storing a reference to the main window ensures that it is easily accessible throughout the entire test.

   For example, type:
   ```
   private Shell eclipse;
   ```

5. Create a `before` method.

   Silk4J also uses this information to connect to the application that you want to test. If the application that you want to test is not running, Silk4J uses this information to launch the application.

   a) Add a `setup` method.
   b) Define the executable name for the application that you want to test.
      For example, to launch and connect to the Eclipse application, type:
      ```
      String executable = "eclipse.exe";
      ```
   c) Add the command line arguments.
      For example, to test the Eclipse IDE, type:
      ```
      String commandLineArgs = "-data " + WORKSPACE_LOCATION;
      ```
      The *WORKSPACE_LOCATION* is defined as:
      ```
      private static final String WORKSPACE_LOCATION = "C:/temp/demo_workspace";
      ```
   d) Add the locator string that describes the main window of the application that you want to test and the technology type of the application that you want to test. Optionally, add a working directory.
      For example, to test the Eclipse IDE, type:
      ```
      eclipse = (Shell)desktop.executeBaseState(executable, commandLineArgs,
          null, "/Shell[@caption='*Eclipse*']", TechDomain.SWT);
      ```
      The XPath locator string is `"/Shell[@caption='*Eclipse*']"`

      `"SWT"` is the technology type that the Eclipse IDE uses for testing.

      `null` represents that no working directory is specified.

      The DefaultBaseState will look for a Shell with a caption that matches `Eclipse`.

6. Close any dialogs that display when the application that you want to test starts.
   For example, when a new workspace is launched in Eclipse, a **Welcome** dialog displays. To close the **Welcome** dialog if it displays, type:

```
String welcomeTabLocator = ".//CTabItem[@caption = 'Welcome']";
if(eclipse.exists(welcomeTabLocator))
    ((CTabItem)eclipse.find(welcomeTabLocator)).close();
```

7. Test the base state.
   Create an empty test method and run the test.

```
@Test
public void emptyTest() {
}
```

   If the test fails, an exception is thrown.

8. Choose **Source ➤ Organize Imports** to have Eclipse automatically add and update all the required import packages to include.

At this point, the test class for the Eclipse IDE example looks like:

```
package com.borland.demo;

public class DynamicObjectRecognitionDemo {
    private static final String WORKSPACE_LOCATION = "C:/temp/demo_workspace";

    private final Desktop desktop = new Desktop();

    private Shell eclipse;

    @Before
    public void setUp() {

        // setup the base state
        String executable = "eclipse.exe";
        String commandLineArgs = "-data " + WORKSPACE_LOCATION;
        eclipse = (Shell)desktop.executeBaseState(executable,
            commandLineArgs, null, "/Shell[@caption='*Eclipse*']",
            TechDomain.SWT);

        // close the welcome screen, if open
        String welcomeTabLocator = ".//CTabItem[@caption = 'Welcome']";
        if(eclipse.exists(welcomeTabLocator))
            ((CTabItem)eclipse.find(welcomeTabLocator)).close();
    }
}
```

Related Topics

- *Creating a Test Case for the Getting Started Tutorial* on page 7

Creating a Test Method for the Getting Started Tutorial

Use the Java perspective within Eclipse to perform this task.

This task illustrates how to create a test method using the example of creating a new Java project and a new Java class for an Eclipse IDE. The test verifies that the Java class is available in the Package Explorer and then the newly created project is deleted.

1. Specify the project name, package name, and class name to which you want to add the test method.
2. Annotate the test method with `@Test` and add the test script.
   For example, you might type:

```
private static final String PROJECT_NAME = "helloWorldProject";
private static final String PACKAGE_NAME = "com.example";
private static final String CLASS_NAME = "HelloWorldClass";

    @Test
    public void testCreateJavaClass() {
        try {
            createProject(PROJECT_NAME);
            createNewJavaClass(CLASS_NAME, PACKAGE_NAME);
            verifyProjectTree();
        } finally {
            deleteProject(PROJECT_NAME);
    }
}
```

3. Implement the method to create a project.

   In our example, the `create project` method opens the **New Java Project** dialog. You can use **File ➤ New ➤ Java Project** to do this.

```
private void createProject(String name) {
    Shell newProjectDialog = openNewJavaProjectDialog(eclipse);
    TextField projectName = (TextField) newProjectDialog.find(".//TextField");

    projectName.setText(name);
    ((PushButton)newProjectDialog.find(
        ".//PushButton[@caption ='Finish']")).select();
    waitForDialogClose(newProjectDialog);
}

private Shell openNewJavaProjectDialog(Shell eclipse) {
    ((Menu)eclipse.find(".//Menu[@caption='File']")).click();
    ((MenuItem)eclipse.find(".//MenuItem[@caption='New*']")).click();
    ((MenuItem)eclipse.find(".//MenuItem[@caption='Java Project']")).click();
    return (Shell) eclipse.find("./Shell[@caption = 'New Java Project']");
}
```

4. Create the method to create a Java class.

   To open the **New Java class** dialog, use **File ➤ New ➤ Class**. After the **New Java class** dialog opens, enter a package name and a class name and click the **Finish** button.

```
private void createNewJavaClass(String name, String thePackage) {
    TestObject newJavaClassDialog = openNewJavaClassDialog(eclipse);
    TextField packageName = (TextField)newJavaClassDialog.find(
        "//TextField[2]");
    packageName.setText(thePackage);
    TextField className =(TextField)newJavaClassDialog.find("//TextField[4]");

    className.setText(name);
    PushButton finishButton = ((PushButton)newJavaClassDialog.find(
        ".//PushButton[@caption = Finish']"));
    finishButton.select();
    waitForDialogClose(newJavaClassDialog);
}

private void waitForDialogClose(TestObject dialog) {
    while(dialog.exists())
```

```
        Utils.sleep(500);
}

private Shell openNewJavaClassDialog(Shell eclipse) {
    ((Menu)eclipse.find(".//Menu[@caption='File']")).click();
    ((MenuItem)eclipse.find(".//MenuItem[@caption='New*']")).click();
    ((MenuItem)eclipse.find(".//MenuItem[@caption='Class']")).click();
    return (Shell) eclipse.find("./Shell[@caption = 'New Java Class']");
}
```

5. Verify that the Java class was created.

   The content of a tab item is not a child of the tab item. This means that there is no reliable way of locating a view's content if the content is not unique. In the case of the Package Explorer view, you must iterate over all trees in the Eclipse application and find the tree with your project. Once you have the correct tree, verify that the class was created by locating the corresponding item in the tree.

   👉 **Note:** If you are testing a Java SWT application and have included custom attributes within your application, you can use the custom attributes to easily locate controls in your application. Using custom attributes to verify the Java class eliminates the need to iterate through the trees and speeds the verification step.

```
private void verifyProjectTree() {
    ItemPath classItem = asItemPath("/" + PROJECT_NAME + "/src/" +
        PACKAGE_NAME + "/" + CLASS_NAME + ".java");
    SWTTree packageTree = getPackageTree();
    Assert.assertNotNull("No tree with the project '"+ PROJECT_NAME
        + "' was found.", packageTree);
    Assert.assertTrue("Class was not created",
        packageTree.getItemPaths().contains(classItem));
}

private SWTTree getPackageTree() {
    ItemPath projectItem = asItemPath("/" + PROJECT_NAME);
    List<TestObject> trees = eclipse.findAll(".//SWTTree");
    for (TestObject tree : trees) {
        if (((SWTTree)tree).getItemPaths().contains(projectItem)) {
            return (SWTTree)tree;
        }
    }
    return null;
}
```

6. Verify that the test is repeatable.

   To verify that the test is repeatable, perform a cleanup. Otherwise, you will not be able to create the Java project in the next test run if it already exists. Implement the `delete project` method, which locates the project in the Package Explorer view and deletes it. When deleting the project you are prompted whether to delete it also from the hard disk. It is important to delete the project from the hard disk, otherwise the next creation will fail.

```
private void deleteProject(String projectName) {
    SWTTree packageTree = getPackageTree();
    if (packageTree != null) {
        ItemPath projectItem = asItemPath("/"+ PROJECT_NAME);
        packageTree.select(projectItem);
        packageTree.typeKeys("<Delete>");
        Shell confirmDeleteDialog = (Shell) desktop.find(
            "//Shell[@caption='Delete Resources']");
        ((CheckBox) confirmDeleteDialog.find(
            ".//CheckBox[@caption='*disk*']")).select(1);
```

```
        ((PushButton) confirmDeleteDialog.find(
            ".//PushButton[@caption='OK']")).select();
        waitForDialogClose(confirmDeleteDialog);
    }
}
```

The entire Eclipse IDE example results in the following class:

```
package com.borland.demo;

import java.util.List;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import com.borland.silktest.jtf.CheckBox;
import com.borland.silktest.jtf.Desktop;
import com.borland.silktest.jtf.Menu;
import com.borland.silktest.jtf.MenuItem;
import com.borland.silktest.jtf.PushButton;
import com.borland.silktest.jtf.TechDomain;
import com.borland.silktest.jtf.TestObject;
import com.borland.silktest.jtf.TextField;
import com.borland.silktest.jtf.Utils;
import com.borland.silktest.jtf.common.types.ItemPath;
import com.borland.silktest.jtf.swt.CTabItem;
import com.borland.silktest.jtf.swt.SWTTree;
import com.borland.silktest.jtf.swt.Shell;

public class DynamicObjectRecognitionDemo {
    private static final String PACKAGE_NAME = "com.example";
    private static final String PROJECT_NAME = "helloWorldProject";
    private static final String CLASS_NAME = "HelloWorldClass";
    private static final String WORKSPACE_LOCATION = "C:/temp/demo_workspace";

    private final Desktop desktop = new Desktop();

    private Shell eclipse;

    @Before
    public void setUp() {
        // setup the base state
        String executable = "eclipse.exe";
        String commandLineArgs = "-data " + WORKSPACE_LOCATION;
        eclipse = (Shell)desktop.executeBaseState(executable,
            commandLineArgs, null, "/Shell[@caption='*Eclipse*']",
            TechDomain.SWT);

        // close the welcome screen, if open
        String welcomeTabLocator = ".//CTabItem[@caption = 'Welcome']";
        if(eclipse.exists(welcomeTabLocator))
            ((CTabItem)eclipse.find(welcomeTabLocator)).close();
    }
}

    @Test
    public void testCreateJavaClass() {
        try {
            createProject(PROJECT_NAME);
            createNewJavaClass(CLASS_NAME, PACKAGE_NAME);
```

```
            verifyProjectTree();
        } finally {
            deleteProject(PROJECT_NAME);
        }
    }

    /**
     * Finds the tree in the package explorer view after our test project
     * has been created. The tree has no unique characteristics and
     * therefore we identify it by its content.
     * @ return the tree in the package explorer view or <code>null
     * </code> if no such tree can be found
     */

    private SWTTree getPackageTree() {
        ItemPath projectItem = asItemPath("/" + PROJECT_NAME);
        List<TestObject> trees = eclipse.findAll(".//SWTTree");
        for (TestObject tree : trees) {
            if (((SWTTree)tree).getItemPaths().contains(projectItem)) {
                return (SWTTree)tree;
            }
        }
        return null;
    }

    private void createProject(String name) {
        Shell newProjectDialog = openNewJavaProjectDialog(eclipse);
        TextField projectName = (TextField) newProjectDialog.find(
            ".//TextField");
        projectName.setText(name);
        ((PushButton)newProjectDialog.find(
            ".//PushButton[@caption ='Finish']")).select();
        waitForDialogClose(newProjectDialog);
    }

    private void createNewJavaClass(String name, String thePackage) {
        TestObject newJavaClassDialog = openNewJavaClassDialog(eclipse);
        TextField packageName = (TextField)newJavaClassDialog.find(
            "//TextField[2]");
        packageName.setText(thePackage);
        TextField className =(TextField)newJavaClassDialog.find(
            "//TextField[4]");
        className.setText(name);
        PushButton finishButton = ((PushButton)newJavaClassDialog.find(
            ".//PushButton[@caption = Finish']"));
        finishButton.select();
        waitForDialogClose(newJavaClassDialog);
    }

    private void waitForDialogClose(TestObject dialog) {
        while(dialog.exists())
            Utils.sleep(500);
    }

    private Shell openNewJavaProjectDialog(Shell eclipse) {
        ((Menu)eclipse.find(".//Menu[@caption='File']")).click();
        ((MenuItem)eclipse.find(".//MenuItem[@caption='New*']")).click();
        ((MenuItem)eclipse.find(
            ".//MenuItem[@caption='Java Project']")).click();
        return (Shell) eclipse.find("./Shell[@caption = 'New Java Project']");
    }
```

```
    private Shell openNewJavaClassDialog(Shell eclipse) {
        ((Menu)eclipse.find(".//Menu[@caption='File']")).click();
        ((MenuItem)eclipse.find(".//MenuItem[@caption='New*']")).click();
        ((MenuItem)eclipse.find(".//MenuItem[@caption='Class']")).click();
        return (Shell)eclipse.find("./Shell[@caption='New Java Class']");
    }

    private void verifyProjectTree() {
        ItemPath classItem = asItemPath("/" + PROJECT_NAME + "/src/" +
            PACKAGE_NAME + "/" + CLASS_NAME + ".java");
        SWTTree packageTree = getPackageTree();
        Assert.assertNotNull("No tree with the project '"+
            PROJECT_NAME + "' was found.", packageTree);
        Assert.assertTrue("Class was not created",
           packageTree.getItemPaths().contains(classItem));
    }

    private void deleteProject(String projectName) {
        SWTTree packageTree = getPackageTree();
        if (packageTree != null) {
            ItemPath projectItem = asItemPath("/"+ PROJECT_NAME);
            packageTree.select(projectItem);
            packageTree.typeKeys("<Delete>");
            Shell confirmDeleteDialog = (Shell) desktop.find(
                "//Shell[@caption='Delete Resources']");
            ((CheckBox) confirmDeleteDialog.find(
                ".//CheckBox[@caption='*disk*']")).select(1);
            ((PushButton) confirmDeleteDialog.find(
                ".//PushButton[@caption='OK']")).select();
            waitForDialogClose(confirmDeleteDialog);
        }
    }
}
```

👉 **Note:** You can access this script in the `samples` directory located at: `<SilkTest_install_directory>\ng\samples\java\swt\Eclipse (Silk4J)`.

Related Topics

- *Creating a Test Case for the Getting Started Tutorial* on page 7

Running Test Cases

Choose one of the following methods to run a test case.

Related Topics

- *Getting Started Tutorial* on page 6
- *Running a Test Case from Eclipse* on page 16
- *Running a Test Case from the Command Line* on page 16

Running a Test Case from Eclipse

Before you run a test case, ensure that the SilkTest Open Agent is running.

1. Navigate to the test case that you want to test.
2. Choose one of the following:
   - Right-click the test class package name to run all tests in the project.
   - Right-click the test class class name to run a test for only that class.
3. Choose **Run As ➤ JUnit Test.**
   The test results display in the JUnit view as the test runs. If all tests pass, the status bar is green. If one or more tests fail, the status bar is red. You can click a failed test to display the stack trace in the Failure Trace area.

Related Topics

- *Running Test Cases* on page 15

Running a Test Case from the Command Line

You must update the PATH variable to reference your JDK location before performing this task. For details, reference the Sun documentation at: *http://java.sun.com/j2se/1.5.0/install-windows.html*.

1. Set the CLASSPATH to:

```
set CLASSPATH=<eclipse_install_directory>\plugins\org.junit4_4.3.1\junit.jar;
<silktest_install_directory>\ng\Silk4J\eclipse\plugins\
com.borland.silktest.jtf_9.2.0.buildnumber\JTF\silktest-jtf-nodeps.jar;
C:\myproject\
```

2. Run the JUnit test case by typing:

```
java org.junit.runner.JUnitCore <test class name>
```

   Note: For troubleshooting information, reference the JUnit documentation at: *http://junit.sourceforge.net/doc/faq/faq.htm#running_1*.

Related Topics

- *Running Test Cases* on page 15

Index