Borland®

THE OPEN ALM COMPANY

SilkTest® 2008 R2

Silk4J Quick Start Tutorial for Dynamic Object Recognition

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

Contents

Quick Start Tutorial

This tutorial provides a step-by-step introduction to using Silk4J to test a web application using dynamic object recognition. Dynamic object recognition enables you to write test cases that use XPath queries to find and identify objects. Dynamic object recognition uses a `Find` or `FindAll` method to identify an object in a test case.

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

For the sake of simplicity, this guide assumes that you have installed Silk4J and are using the GMO sample web application.

For additional information about Silk4J, including information about sample scripts and applications, 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 Silk4J

1. Choose **Start ➤ Programs ➤ Borland ➤ SilkTest <version> ➤ Eclipse (with Silk4J)**.
   If you have not specified a default workspace location or if this is the first time that you are launching Silk4J, the **Workspace Launcher** dialog box opens.
2. Specify the location of your workspace and click **OK**.
   Eclipse opens the **Welcome** view the first time you launch Eclipse.

Related Topics

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

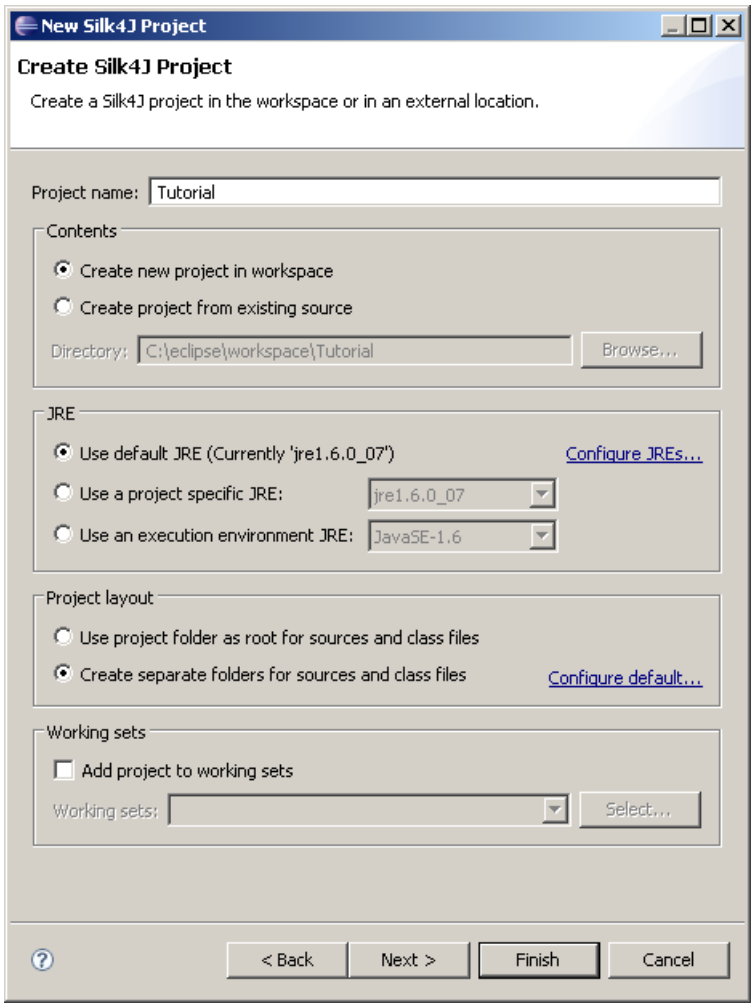
- *Quick Start Tutorial* on page 4

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

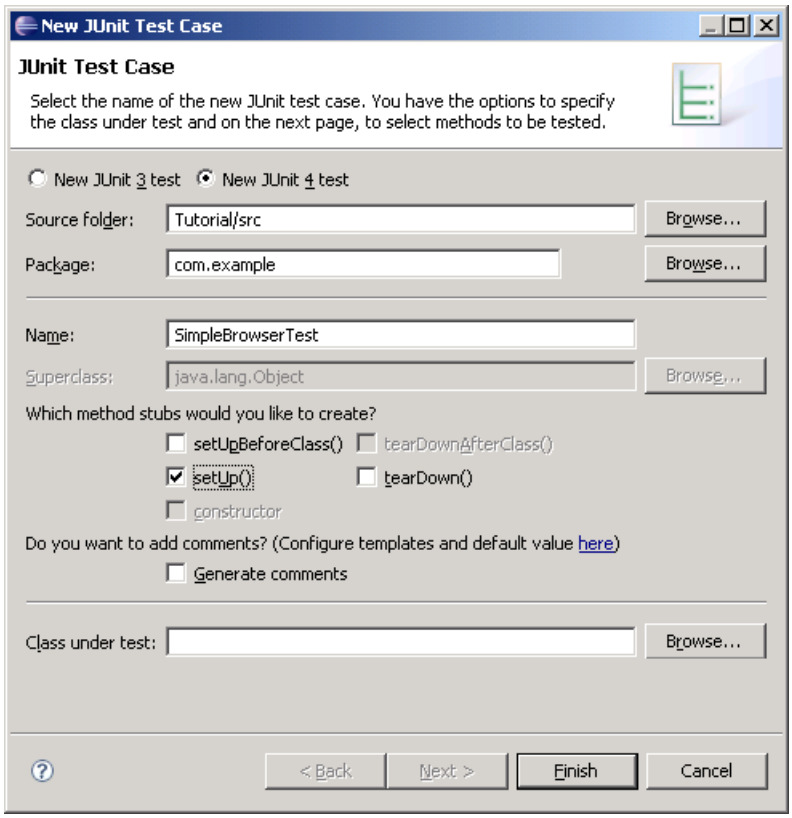
- *Quick Start Tutorial* on page 4

Creating the JUnit Test Case for the Quick Start 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.

For example, type: `com.example`.

4. In the **Name** text box, specify the name for the test case.
   For example, type: `SimpleBrowserTest`.
5. Check the **setUp ()** check box.



6. Click **Finish**.
   The new class file opens.

The new class contains the following code:

```
package com.example;

import org.junit.Before;

public class SimpleBrowserTest {

    @Before
    public void setUp() throws Exception {
    }

}
```

Related Topics

- *Quick Start Tutorial* on page 4

Creating the Setup Method

Use the `setup` method to prepare the application that you want to test and bring it to a defined starting point.

👉 **Note:** The `setup` method is called before every test method in this class.

1. Add `private Desktop desktop = new Desktop();` as the first line in the class.
2. Choose **Source ➤ Organize Imports** to have Eclipse automatically add and update all the required import packages to include.
   The **Organize Imports** dialog box opens if there are multiple Desktop classes available. Otherwise, Eclipse automatically adds and updates all the required import packages to include..
3. If the **Organize Imports** dialog box opens, select `com.borland.silktest.jtf.Desktop` and then click **Finish**.
   Eclipse automatically adds and updates all the required import packages to include.
4. In the `setUp()` method, include an `executeBaseState` method to connect the GMO sample application to Silk4J.

   Because the sample application is a web application, attach to Internet Explorer and specify the xBrowser technology domain.

   Type:

   ```
   desktop.executeBaseState(
       "C:/Program Files/Internet Explorer/iexplore.exe", null, null,
       ".//BrowserWindow", TechDomain.XBROWSER);
   ```

5. To establish a link to the handle for the browser that you will use for testing, perform the following steps:
   a) Add `private BrowserWindow browser;` as the second line in the class.
   b) In the `setUp()` method, assign the browser window in the `executeBaseState` method by adding: `browser = (BrowserWindow).`
   The code looks like the following:

   ```
   browser = (BrowserWindow)desktop.executeBaseState(
       "C:/Program Files/Internet Explorer/iexplore.exe", null, null,
       ".//BrowserWindow", TechDomain.XBROWSER);
   ```

   The handle for the browser window is valid as long as the browser window exists.
6. In the `setUp()` method, add the following line to navigate to the GMO web site:

   ```
   browser.navigate("http://demo.borland.com/gmopost/");
   ```

   The code looks like the following:

   ```
   browser = (BrowserWindow)desktop.executeBaseState(
       "C:/Program Files/Internet Explorer/iexplore.exe", null, null,
       ".//BrowserWindow", TechDomain.XBROWSER);
   browser.navigate("http://demo.borland.com/gmopost/");
   ```

7. Choose **Source ➤ Organize Imports** to have Eclipse automatically add and update all the required import packages to include.
   `import com.borland.silktest.jtf.TechDomain` and `import com.borland.silktest.jtf.xbrowser.BrowserWindow` are added to the import list.
8. Choose **File ➤ Save** to save the method.
   Every time you save, Eclipse compiles the source files. If something is incorrect, Eclipse underlines it with a red line.

Review your work to ensure your test class looks like the following:

```
package com.example;

import org.junit.Before;

import com.borland.silktest.jtf.Desktop;
import com.borland.silktest.jtf.TechDomain;
import com.borland.silktest.jtf.xbrowser.BrowserWindow;

public class SimpleBrowserTest{
    private Desktop desktop = new Desktop();
    private BrowserWindow browser;

    @Before
    public void setUp() throws Exception{
        browser = (BrowserWindow)desktop.executeBaseState(
            "C:/Program Files/Internet Explorer/iexplore.exe", null, null,
            ".//BrowserWindow", TechDomain.XBROWSER);
        browser.navigate("http://demo.borland.com/gmopost/");
    }
}
```

Related Topics

- *Quick Start Tutorial* on page 4

Creating the Test Method for the Quick Start Tutorial

In this test method, we will test clicking the **Enter GMO OnLine** button in the GMO sample application.

1. Open Internet Explorer and navigate to the GMO web site at: *http://demo.borland.com/gmopost/*.
2. In Silk4J, in the `setup` method, type `test`.

   Make sure to place the caret in the `setup` method when you type `test`.
3. Press Ctrl+Space and choose **Test - test method (JUnit4)**.
   Eclipse generates the method stub for the test.
4. Change the method name from `testname()` to `testEnterGMOOnline()`.
5. Find the correct parameter for the `Find` method by using the Silk4J Spy.

   The test case must use the `Find` or `FindAll` method and a query string that identifies the object you want to test. You must use a supported construct to create the query string. Using the Silk4J Spy ensures that the query string is valid.

   For more information about the syntax of XPath query strings, see "Supported XPath Subset" in the *Silk4J User Guide*.

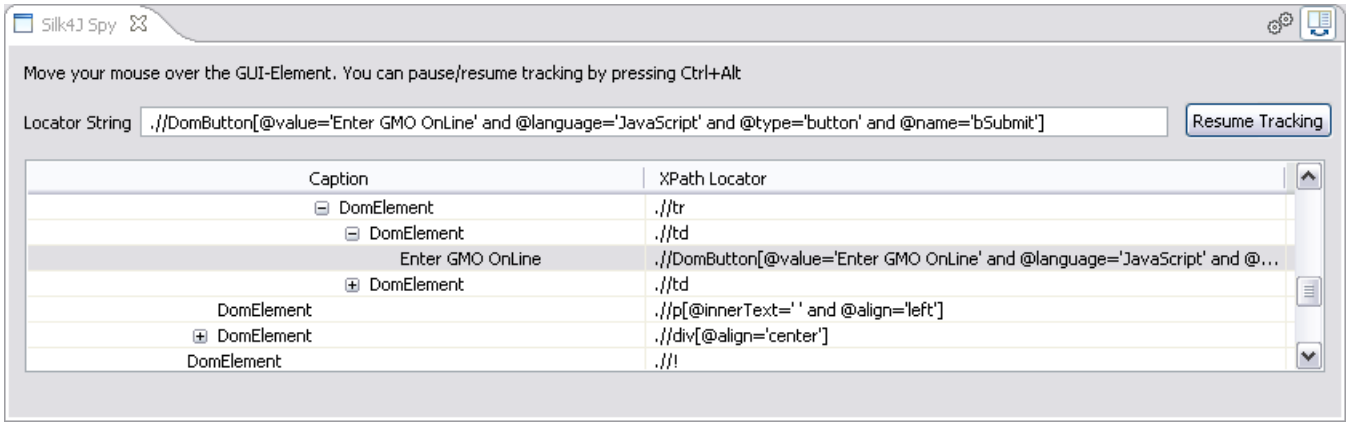
   a) Choose **Window ➤ Show View ➤ Other ➤ Silk4J ➤ Silk4J Spy**.
   b) Click **OK**.
      The Silk4J Spy opens.
   c) In the Views toolbar, click the Attach to Process icon.

      This command activates the application for testing, which was done programmatically in the *Creating the Setup Method* step.

      The **Configure Applications** dialog box opens.

d) In the table, select the Internet Explorer (`iexplore.exe`) process for the Green Mountain Outpost web application.
e) Click **Attach**.
WIN32 and xBrowser display in the **Loaded Tech Domains** column.
f) Click **Close**.
The **Configure Applications** dialog box closes.

6. In the GMO sample application, move your mouse over the **Enter GMO OnLine** button and press Ctrl+Alt. The element identifier is displayed in the **Locator String** text box.



7. Copy the locator string and attributes into the `Find` method in your script.

   When you run the test, Silk4J searches for the object with the assigned attributes specified in the locator string. In this case, Silk4J searches for the DOM button with the value, language, type, and name attributes shown in the code below.

   The code looks like the following:

```
DomButton btn = (DomButton)browser.find(
    ".//DomButton[@value='Enter GMO OnLine' and" +
    "@language='JavaScript' and @type='button' and @name='bSubmit']");
```

8. Remove all the attributes, except the `@name='bSubmit'` attribute.

   In this case, the `name` attribute uniquely identifies the button, so no other attributes are necessary.

   Note: It is a good practice to include only the required attributes in a test method.

9. To include the click button functionality, add: `btn.click();`.
   The code looks like the following:

```
@Test
public void testEnterGMOOnline() throws Exception {
    DomButton btn = (DomButton)browser.find(
        ".//DomButton[@name='bSubmit']");
    btn.click();
}
```

10. Choose **File ➤ Save** to save the method.
    Every time you save, Eclipse compiles the source files. If something is incorrect, Eclipse underlines it with a red line.

Review your work to ensure your test class looks like the following:

```
package com.example;

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

import com.borland.silktest.jtf.Desktop;
import com.borland.silktest.jtf.TechDomain;
import com.borland.silktest.jtf.xbrowser.BrowserWindow;
import com.borland.silktest.jtf.xbrowser.DomButton;

public class SimpleBrowserTest{
    private Desktop desktop = new Desktop();
    private BrowserWindow browser;

    @Before
    public void setUp() throws Exception{
        browser = (BrowserWindow)desktop.executeBaseState(
            "C:/Program Files/Internet Explorer/iexplore.exe", null, null,
            ".//BrowserWindow", TechDomain.XBROWSER);
        browser.navigate("http://demo.borland.com/gmopost/");
    }

    @Test
    public void testEnterGMOOnline() throws Exception {
        DomButton btn = (DomButton)browser.find(
            ".//DomButton[@name='bSubmit']");
        btn.click();
    }
}
```

Related Topics

- *Quick Start Tutorial* on page 4

Running the Test Method

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

Right-click the **testenterGMOOnline** test method in the Package Explorer and choose **Run As ➤ JUnit Test**.

The test results display in the JUnit view as the test runs. If the test passes, the status bar is green. If the test fails, the status bar is red. You can click a failed test to display the stack trace in the Failure Trace area.

Related Topics

- *Quick Start Tutorial* on page 4

Index