Unit test portlets with portletUnit

Use the portletUnit framework to test without portal servers

Skill Level: Introductory

Hema Venkatrangan (hema.sudarshan@in.ibm.com)
System Engineer
IBM Global Services

20 Apr 2010

Portals provide information from diverse sources in a unified way. When portlets are integrated into a portal, the functions delivered as part of each portlet should be available all of the time. Unit testing can ensure that the features of your portlets will work all the time, and everywhere. It is important to unit test portlets before they are hosted publicly, and testing with frameworks will speed up the testing process. In this article, learn about portletUnit, a JUnit testing framework for testing JSR-168 portlets. portletUnit uses a mock container for testing, thereby reducing the cost of investment in huge portal server environments. With the portletUnit framework you can run unit tests on any machine—without a portal server.

Introduction

portletUnit is a Java™ Unit (JUnit) Testing framework for testing JSR-168 portlets outside a portlet container (just as servletUnit is used to test servlets outside a servlet container). portletUnit is built on servletUnit and Pluto. In this article, learn how to use the open source test framework portletUnit to unit test portlet applications.

Download the sample code with this article to configure portletUnit in your workspace, write unit tests, and test a portlet.

With the portletUnit framework, you can test the UI of a portlet with actions related to the controls within the portlet. portletUnit gives you the capability to do unit tests on code and to execute the portlets without deploying them to a server. portletUnit is

© Copyright IBM Corporation 2010. All rights reserved.

built on top of httpUnit, which helps in testing a UI rendered in a Web page. The tests tend to run on mock portlet containers, so you don't need to start the portal server environment. Using mock containers reduces your testing time without compromising the portal server functions. The portletUnit framework also comes with test harness classes to test different modes of a portlet. You can use this framework to test the complete life cycle of a portlet.

Prerequisites and assumptions

To follow along with the examples in this article, you should be familiar with:

- Using Eclipse-related IDEs; the example in this article was developed using RSA
- Portal development
- The basics of using JUnit

It is assumed that you know:

- How to develop portlets
- How to create JUnit test cases
- A bit about mock objects

In the example in this article, both the portlet application and the JUnit test cases project reside in the same workspace. If the portlet application resides in a different workspace, it could still be tested. portletUnit requires the full path for the portlet application to the WebContent folder.

Why you need unit testing and unit testing frameworks

As the saying goes, "If a program feature lacks an automated test, we assume it doesn't work." The goal of unit testing is to verify whether the smallest unit of code behaves the way it is expected to. Unit testing defines a written contract about the behavior of each unit. Each unit, defined at the method level, is tested separately before integration with its respective modules. Unit testing is the safe way to ensure that your feature will work today and forever. Testing can prove whether the application can run effectively in any environment. If it can't, the application has to be redesigned based on requirements.

Given the number of units to be tested, a manual approach to testing step-by-step is hard to imagine. An automated approach can easily achieve the goal in a more controlled way. To control the execution of testing, and to automate the testing approach, you need a framework. As an added benefit, a testing framework also

© Copyright IBM Corporation 2010. All rights reserved.

forces you to explicitly declare the expected results of specific program execution routes. After writing a test that expresses the result you're trying to achieve, you then debug until the test is positive. With a set of tests for all the core components of the project, you can modify specific areas and immediately see the effect of the modifications on other areas. With such test results, you'll quickly see any side effects.

JUnit, a simple framework for writing and running automated tests, will help you develop robust tests. When faced with unit testing, many teams end up producing some type of testing framework. JUnit, available as open source, eliminates this onerous task by providing a ready-made framework. JUnit is:

- Best used as an integral part of a development testing regime; it provides a mechanism to consistently write and execute tests
- A framework for implementing testing in Java that provides a simple way to explicitly test specific areas of a Java program
- Extensible and can be used to test a hierarchy of program code either singularly or as multiple units

Unit testing your portlets

All enterprise applications require complete testing since they'll be deployed on different application servers. A portlet is an enterprise application, and testing portlets is required. You need to be sure that each and every feature of the portlet works as expected.

The unit testing of portlets should cover the functional testing for each mode of the portlet *and* the actions performed on the portlets. The testing should basically test the portal server at the HTTP level. The unit testing on portlets should help test the following functions:

- A proper request is sent and the corresponding response is obtained
- Expected form elements are present in a page
- Expected process actions are available on each page
- Navigation from one page to another
- Result of process action (`processAction` is the method in a portlet that deals with actions performed on a page.)

To test applications you need containers, or the role of containers. Containers, such as J2EE containers, are the runtime environment of the enterprise applications. Running unit tests on different workstations requires application servers where the containers can run. To execute automated testing on dedicated machines also requires application servers. There are many unit testing frameworks available today that use mock containers for testing. These testing frameworks help reduce the cost of investing in huge application servers. By using testing frameworks you can run tests on any machine—without even a simple application server.

The portletUnit framework

portletUnit is a unit testing framework to test portlets. It allows automated testing of Web applications. The tests are written in Java code. portletUnit is used to test JSR-168-compliant portlets outside of a portlet container. portletUnit is architected to map functions similar to servletUnit onto portlets. portletUnit is built on servletUnit and Pluto; Pluto is the reference implementation of the Java Portlet Specification. The current version of this specification is JSR-168. It provides a mock portlet container, just as servletUnit provides a mock servlet container.

There are many unit testing frameworks available for servlet-based applications. For portlets, only one is currently in use: portletUnit. Though portletUnit is in the initial stages of development, most of the features are covered and can be leveraged to test portlets.

portletUnit and JUnit

As mentioned, JUnit is the de facto standard to test an application developed in Java code. Nevertheless, test-driven portal application development requires broadening the unit test approach. Portal application development has special requirements, so both the scope of unit testing and the JUnit framework itself have to be extended.

The base framework for portletUnit is JUnit. On top of JUnit, many Web application and portal application-related frameworks are integrated to test the application. This article outlines an approach for customizing the unit test cycle and the JUnit framework to achieve test-driven portal application development.

As a testing tool, portletUnit is primarily designed for "black box" testing of portlets. In many cases, that might be all you need. If you're developing complex portlets, though, you may want to test smaller pieces of your code. Sometimes you can isolate portions into simple tests using only JUnit. In other cases, you'll want to test in a portlet environment. At this point you can use a mock portlet container to test the portlets.

The test cases in portletUnit are similar to JUnit, but are embedded with multiple testing frameworks to handle Web-related tasks, server-side tasks, and so on. portletUnit uses JUnit to provide the outline of a testing framework (such as creation of TestCase and TestSuite using assertions). It uses HttpUnit to handle the testing of UI and Web page tasks.

The next section focuses on setting up portletUnit to prepare for unit testing.

Unit testing using portletUnit and JUnit

The unit tests to be developed should test the following:

- Whether the View mode of the portlet displayed
- If you can navigate to Edit mode from View mode
- Whether the heading "available_bookmarks" is present
- Whether initially a URL link of IBM is present in View mode (the portlet is displayed in View mode always)
- Various tests in Edit Mode, such as:
  - Are the two text fields to enter name and URL present?
  - What is the response when the *Set* button is clicked?
  - What is the response when the *Reset* button is clicked?

This article covers the unit tests related to testing whether the View mode of the portlet is displayed and whether you can navigate from View mode to Edit mode. The next step is to write the unit tests.

Using the mock container of portletUnit to write unit tests

As mentioned, the framework runs the portlets in a mock container. In the JUnit tests, you need to configure the mock portlet container to run the portlets. Follow the steps below to configure the portletUnit to run portlets and to configure the mechanism to validate whether the View mode of the portlet is displayed. (You should also follow these steps with other test methods that you develop to test the function of your portlets.)

1. Create an instance of File Object that points to the WebContent folder of the portlet project.
   This folder should contain the WEB-INF folder.

2. Since portletUnit provides a mock container to run the designated portlets, you need to create an instance of the `PortletRunner` class from the portletUnit framework, which is a mock portlet container.
   The constructor takes two arguments:

© Copyright IBM Corporation 2010. All rights reserved.

- File objects that point to WebContent of the portlet project
- Portlet name (which should be the name of the portlet as defined in portlet.xml)

3. Create an instance of the type `GenericPortletUser` of the portletUnit framework.
   Any container requires the authentication of the user to use the server. This class is used to create a dummy user for the mock container, which is automatically validated.

   > `PortletRunner`, `GenericPortletUser`, and `WebResponse` are part of portletUnit.

4. Set the user for the portlet runner by calling `setUser(user)` on the `PortletRunner` instance.

5. Invoke `getResponse()` on the instance of `PortletRunner`.
   The method `getResponse()` returns the response of an action performed, such as a button click or submitting the form.

6. Perform the assertion related to the test case. To perform the assertion pertaining to the response obtained, you need to use the `assert` methods from the portletUnit framework. These methods are similar to JUnit assert methods. Examples include:
   - Checking the number of links, buttons, or text fields on the page
   - Obtaining the name of the text field and asserting it on the expected name
   - Obtaining the value of the text field and asserting it on the expected result

The tests covered in the rest of this article show only the fragment of the JUnit for the bookmark application. The full unit testing-related code is supplied in the sample code.

To begin unit testing the sample portlet using portletUnit, you will:

- Create a portlet application
- Configure the portletUnit testing framework
- Write JUnit tests

Create a portlet application

First and foremost, you need to have a portlet for testing. The sample, called *BookmarkPortlet*, allows you to create and remove bookmarks. The bookmark contains two parts:

- Name of the bookmark
- URL of the bookmark

Actually developing the portlet is outside the scope of this article. You can download the code related to the BookmarkPortlet. Let's focus now on unit testing the portlet using portletUnit.

For testing purposes I created a bookmark named *IBM*. Figure 1 shows the View mode of the BookmarkPortlet.

Figure 1. View mode of BookmarkPortlet

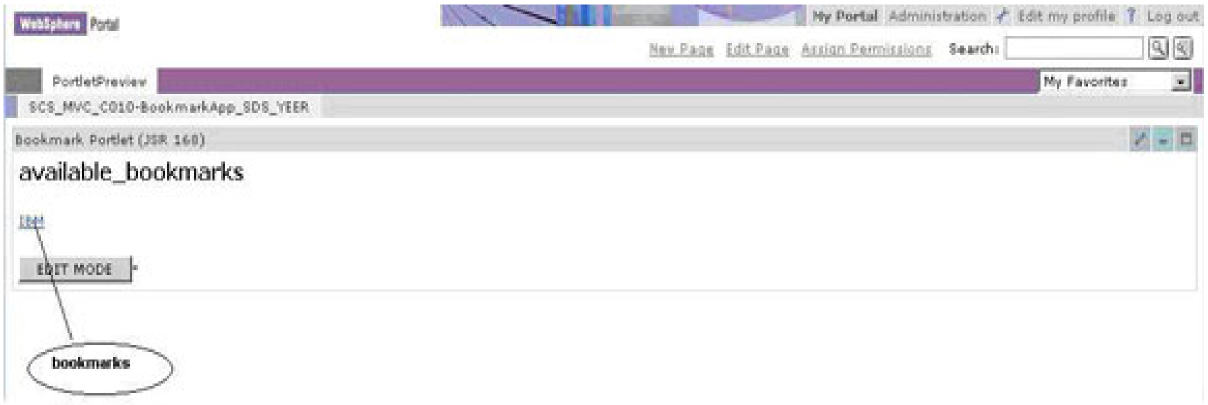
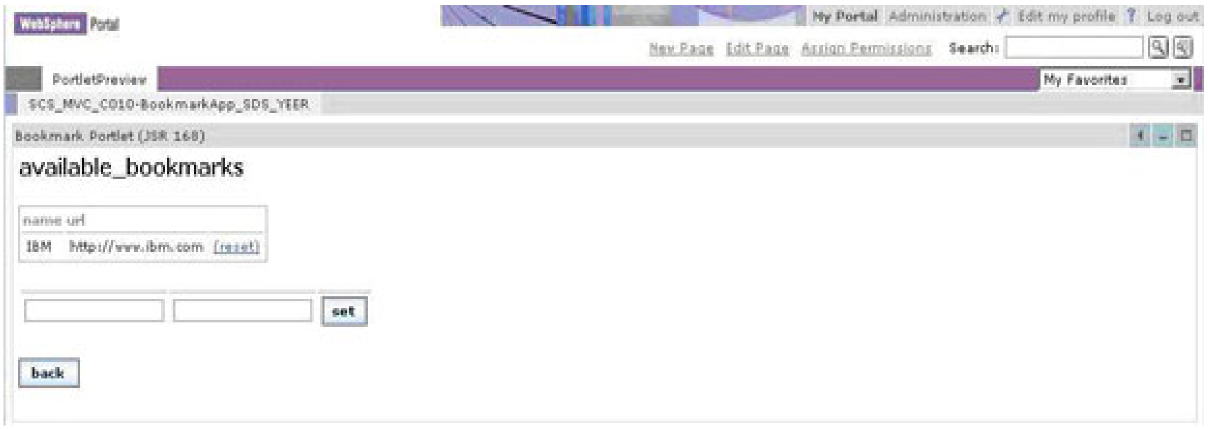


Figure 2 shows the Edit mode of the sample portlet.

Figure 2. Edit mode of BookmarkPortlet



Configure the portletUnit testing framework

To use portletUnit for writing JUnit tests and testing portlets, there are two major steps (each with sub-steps).

1. Create a JUnit project.
    a. In the IDE workspace, where the portlet project exists, create a project of type Java by right clicking on *package explorer* and selecting *New --> Project*.
    b. Call the project `PortletTestUsing_portletUnit.`
2. Download and add all the JARs of the portletUnit framework to the project.
    a. Download the latest distribution from SourceForge (see Resources). The sample in this article uses portletunit-0.2.5.
    b. Right click on the project, and select *Properties --> Java Build path.* Click the **Libraries** tab, then select *Add External jars*.
    c. Add the portletUnit.jar.
    d. Add the following list of dependent JARs required for portletUnit (The JARs are part of the portletUnit distribution zip file.):
        - Tidy.jar
        - httpunit.jar
        - js.jar
        - nekohtml.jar
        - servlet.jar
        - xercesImpl.jar
        - xmlParserAPIs.jar
        - servletunit.jar
        - commons-collections-3.1.jar
        - commons-logging-api.jar
        - junit.jar

© Copyright IBM Corporation 2010. All rights reserved.

- castor-0.9.5.jar
- jasper-compiler.jar
- jasper-runtime.jar
- pluto-1.0.1-rc1.jar
- plutoImpl.jar
- portlet-api-1.0.jar

Write JUnit tests

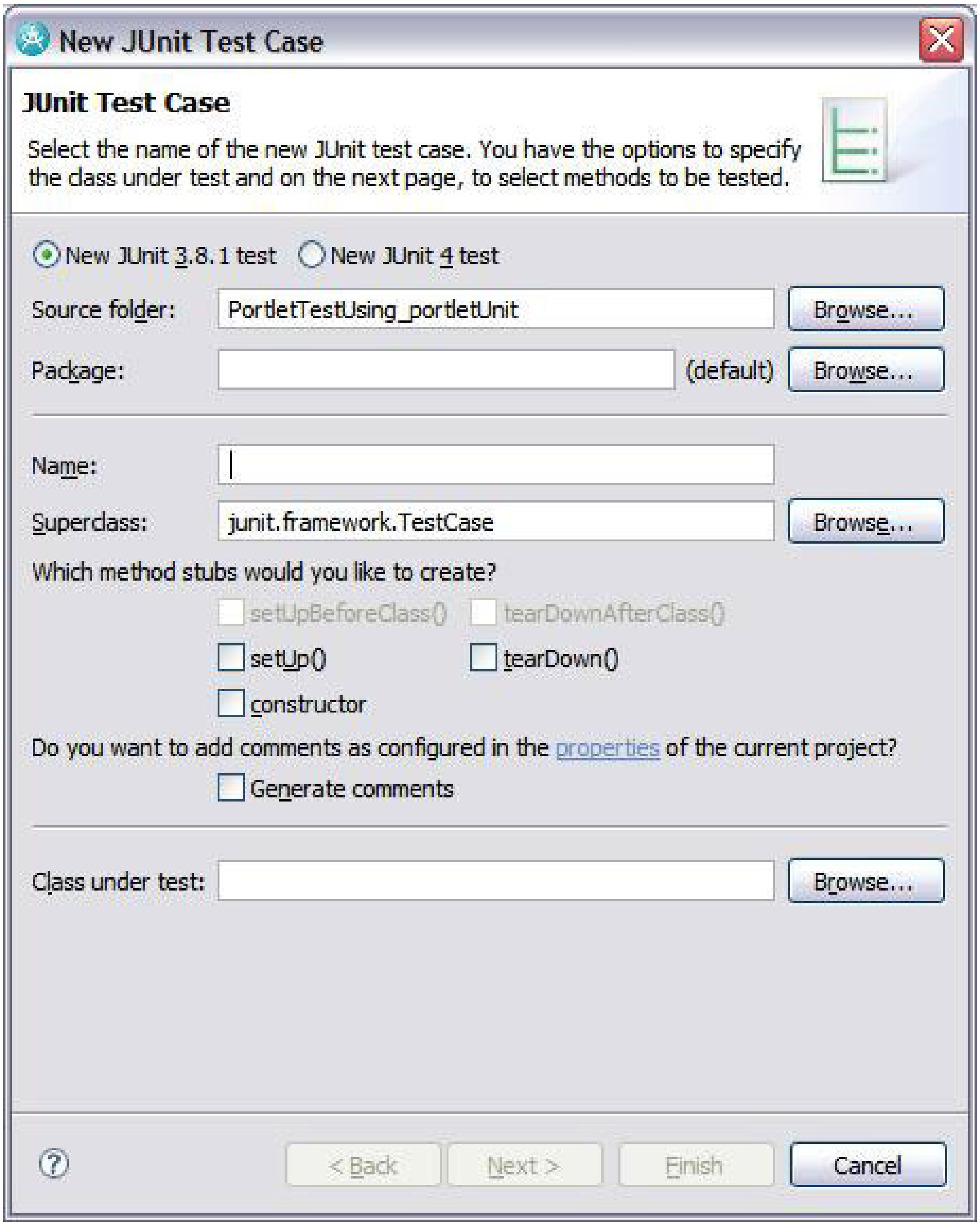
To test the portlet's function, the first step is to create JUnit test cases.

1. In the IDE workspace, where the portlet project exists, right click on the *package explorer*.
   Select *New --> JUnit Test Case*.

2. In the window shown in Figure 3, provide the name of the class, BookmarkExampleTest, and click **Finish**.

Figure 3. Create JUnit test case in IDE

© Copyright IBM Corporation 2010. All rights reserved.


New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

[✓ New JUnit 3.8.1 test] New JUnit 4 test

Source folder: PortletTestUsing_portletUnit

Package: (default)

Name:

Superclass: junit.framework.TestCase

Which method stubs would you like to create?

- [ ] setUpBeforeClass()
- [ ] tearDownAfterClass()
- [ ] setUp()
- [ ] tearDown()
- [ ] constructor

Do you want to add comments as configured in the properties of the current project?

- [ ] Generate comments

Class under test:

Test Case 1: Testing the View mode of the portlet

To write the unit test for testing the View mode of the portlet in the `BookmarkExampleTest` class:

1. Create a method named `testdoViewOfGenericBookmarkWithPortletUnit` whose responsibility is to test the View mode of the portlet.

© Copyright IBM Corporation 2010. All rights reserved.

2. Create an instance of `File` pointing to the WebContent folder of the portlet project, as shown below.

```
File webInfDir=new File(
        "{YourRADWorkspaceforPortlet}/{PortletProject}/WebContent");
```

3. Create a `PortletRunner` instance of the portletUnit framework, which creates the mock portlet container.

```
PortletRunner runner = PortletRunner.createPortletRunner(
        BookmarkGenericPortlet.class,webInfDir, "BookmarkGenericPortlet");
```

   The first argument of the method `createPortletRunner` is the portlet class of your portlet application. The third argument of `createPortletRunner` is the name of the portlet that is defined in portlet.xml.

4. Create a user of type `GenericPortletUser` for the authentication of a user to use the server.

```
PortletUser user = new GenericPortletUser();
```

5. Set the user for the portlet runner by calling `setUser(user)` on the `PortletRunner` instance.

```
runner.setUser(user);
```

6. Invoke `getResponse()` on the instance of `PortletRunner`.

```
WebResponse bookmarkResponse=runner.getResponse();
```

7. Using the WebResponse of any action, you can test any action of the portlet and its functions. From the response object you can obtain any element(s) of the page, such as links, applets, form, select, check box, input types, tables, images, and so on.

© Copyright IBM Corporation 2010. All rights reserved.

From the response object you can assert whether the portlet has been displayed as expected in the View mode. The sample portlet contains one hyperlink named IBM, so from the response object you should obtain the links in the page. For the sample portlet, the number of links should be one, and the link name should be IBM.

```
WebLink[] links=bookmarkResponse.getLinks();
assertEquals(links.length,1);
assertEquals(links[0].getText(),"IBM");
```

Listing 1 shows a code snippet of the method to test the View mode.

Listing 1. Testing View mode of the bookmark portlet

```
public void testdoViewOfGenericBookmarkWithPortletUnit() throws Exception
{
        File webInfDir=new File(
                "{YourRADWorkspaceforPortlet}/{PortletProject}/WebContent");
        Class portletClass = BookmarkGenericPortlet.class;
        PortletRunner runner = PortletRunner.createPortletRunner(
                portletClass,webInfDir, "BookmarkGenericPortlet");
        PortletUser user = new GenericPortletUser();
        runner.setUser(user);
        WebResponse bookmarkResponse=runner.getResponse();

        WebLink[] links=bookmarkResponse.getLinks();
        assertEquals(links.length,1);
        assertEquals(links[0].getText(),"IBM");
}
```

You can enhance this test to cover the assertion of other display parts of the portlet.

Test Case 2: Navigating to Edit mode from View mode

One drawback of portletUnit is that it does not provide operations to switch between modes of a portlet. To switch to other modes, you can tweak at the portlet level. I provided a button in the portlet page to switch from View mode to Edit mode. portletUnit version 0.2.3 didn't have the support to change the code to support this functional testing. I hope future versions of portletUnit will support the feature to test the switching of modes.

Listing 2 shows the code fragment in JSP for navigating to Edit mode.

Listing 2. Testing Edit mode of the bookmark portlet

```
<%
```

© Copyright IBM Corporation 2010. All rights reserved.

```
String editURL=(String)request.getAttribute("edit");
 %>
<form action="<%=editURL%>" method="post"><input name="edit"
        type="submit" value="EDIT MODE"/>
```

To unit test the navigation from View mode to Edit mode, follow the steps defined earlier. Apart from that, the test should perform the execution process of navigation from View mode to Edit mode and verify whether the Edit mode of the portlet is displayed.

1. Create a method named `testClickingEditButtonAndCheckSetURLWithPortletUnit`, whose responsibility is to test the Edit mode of the portlet.
2. Include the steps defined in Test Case 1 to obtain the runner and get the response.
3. Take the portlet to Edit mode.

```
WebForm[] forms=bookmarkResponse.getForms();
assertEquals(forms.length,1);
WebResponse editResponse=forms[0].submit();
```

   In the code above you're doing the form submit to take the portlet to Edit mode. Remember, we added an Edit button in the portlet page to support testing of the Edit mode.

4. Assert the components and actions in Edit mode. When navigating to Edit mode, the portlet page should contain three buttons and two text controls. The buttons are *Set*, *Reset*, and *Back to View mode*; the text controls are to accept the Link name and the URL of the link.
   a. Assert for a button named Set,

```
assertTrue(editForms[j].hasParameterNamed("set"));
```

      where *`j`* corresponds to one of the forms in the Edit mode of the portlet.

   b. Set the values for the text controls and submit the form with the proper action,

```
editForms[j].setParameter("arg0_name", "Yahoo");
```

© Copyright IBM Corporation 2010. All rights reserved.

```
editForms[j].setParameter("arg0_url", "www.yahoo.com");
SubmitButton setButton=editForms[j].getSubmitButton("set");
setButton.click();
```

where:
`set` is the action in our sample portlet, which adds the new url.
`"arg0_name"` is the text control name to accept the name of the link.
`"arg0_url"` is the text control name to accept the URL of the link.

5. Check whether the new link is added. To do this, you need to get the response object of the form submitted, obtain the links, and assert it against a value of 2, which includes a link representing IBM and a link that was just added.

```
WebResponse setResponse=editForms[j].submit(setButton);
WebLink webLinks[]=setResponse.getLinks();
assertEquals(webLinks.length,2);
```

Listing 3 shows the code snippet for testing whether the portlet is navigated to Edit mode.

Listing 3. Testing Edit mode of the bookmark portlet

```
public void testClickingEditButtonAndCheckSetURLWithPortletUnit() throws Exception
{
        File webInfDir=new File(
                "{YourRADWorkspaceforPortlet}/{PortletProject}/WebContent");
        Class portletClass = BookmarkGenericPortlet.class;
        PortletRunner runner = PortletRunner.createPortletRunner(
                portletClass,webInfDir, "BookmarkGenericPortlet");
        PortletUser user = new GenericPortletUser();
        runner.setUser(user);
        WebResponse bookmarkResponse=runner.getResponse();
        WebLink[] links=bookmarkResponse.getLinks();

        assertEquals(links.length,1);
        assertEquals(links[0].getText(),"IBM");

        WebForm[] forms=bookmarkResponse.getForms();
        assertEquals(forms.length,1);
        WebResponse editResponse=forms[0].submit();

        WebForm editForms[]=editResponse.getForms();
        assertEquals(editForms.length,1);

        for (int j=0; j<editForms.length; j++) {
                if(editForms[j].getAction().endsWith("set"))
                {
                assertTrue(editForms[j].hasParameterNamed("arg0_name"));
                assertTrue(editForms[j].hasParameterNamed("arg0_url"));
                editForms[j].setParameter("arg0_name", "Yahoo");
                editForms[j].setParameter("arg0_url", "www.yahoo.com");
                assertTrue(editForms[j].hasParameterNamed("set"));
                SubmitButton setButton=editForms[j].getSubmitButton("set");
                setButton.click();
                WebResponse setResponse=editForms[j].submit(setButton);
                WebLink webLinks[]=setResponse.getLinks();
                assertEquals(webLinks.length,2);
                }
        }
}
```

Conclusion

Unit testing is an integral part of development activities; it ensures that your feature works today and in the future. Performing unit tests on your portlets before they're exposed publicly is a must. In this article, you learned how to use portletUnit, a unit testing framework, to write unit tests and execute them very simply—without portal servers. Because you use mock containers for testing, portletUnit reduces the cost of investing in portal server environments.

© Copyright IBM Corporation 2010. All rights reserved.

Downloads

Description	Name	Size	Download method
Sample code[1]	PortletUnitTest.zip	380KB	HTTP

Information about download methods

Note

1. The sample code contains the Bookmark portlet and portletUnit test cases to test the portlet.

© Copyright IBM Corporation 2010. All rights reserved.

Resources

Learn

- Learn more about Java Portlet Specification (JSR-168), which defines a set of APIs for portal computing addressing the areas of aggregation, personalization, presentation, and security.
- To develop portals using Rational Application Developer, read the IBM Redbook, *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools* (Aug 2005).
- Portlet Development Best Practices and Coding Guidelines (developerWorks, Mar 2003) is a collection of best practices for portlet development using the IBM Portlet API (pre-JSR 168). It provides guidelines for designing and coding portlets for portals powered by IBM WebSphere® Portal.
- Learn all about the JUnit framework.
- Explore the portletUnit framework on SourceForge, the place to find and develop software.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools as well as IT industry trends.
- Follow developerWorks on Twitter.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- The developerWorks Web development zone specializes in articles covering various Web-based solutions.
- The developerWorks Open source zone specializes in extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM products.

Get products and technologies

- Download the portletUnit framework.
- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Hema Venkatrangan

Hema Venkatrangan is part of Global Business Solution Centre (GBSC) in the Common Technical Service area at IBM. She is involved in the design and development of common technical services to be leveraged in other development projects. Hema has nearly six years of IT experience and five years of teaching experience. Her areas of expertise include Java development and portal development.

© Copyright IBM Corporation 2010. All rights reserved.