Junit Testing with Struts (StrutsTestCases extension)

This tutorial gives you an overview about the StrutsTestCases extension of JUnit and example how you can use it to test your Struts application.

General

Author:
Sascha Wolski
Sebastian Hennebrueder
http://www.laliluna.de/tutorials.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
April, 13 2005

Software:
Eclipse 3.x
MyEclipse 3.8.x
StrutsTestCases for JUnit 2.1.3

Source code:
http://www.laliluna.de/assets/tutorials/junit-testing-struts-source-light.zip

The sources do not contain any project files of eclipse or libraries. Create a new project following the tutorial, add the libraries as explained in the tutorial and then you can copy the sources in your new project.
PDF Version
http://www.laliluna.de/assets/tutorials/junit-testing-struts-en.pdf

What is StrutsTestCases ?

StrutsTestCase for JUnit is an extension of the standard JUnit TestCase class that provides capabilities for testing code developed with the Struts framework. StrutsTestCase provides a Mock Object approach and a Cactus approach to actually run the Struts ActionServlet. A Mock Object expects that you are using a running servlet engine. The Cactus approach does not expect a servlet engine.

Because StrutsTestCase uses the ActionServlet controller to test your code, you can test not only the implementation of your Action objects, but also your mappings, form beans, and forwards declarations. And because StrutsTestCase already provides validation methods, it's quick and easy to write unit test cases.

What is the MockStrutsTestCase ?

The MockStrutsTestCase use a set of HttpServlet mock objects to simulate the container environment without running a servlet engine. You can test your Struts code without running a application server or deploying your code to a application server. To use the MockStrutsTestCase you have to extend your test classes from the MockStrutsTestCase class.

What is the CactusStrutsTestCase ?

The CactusStrutsTestCase use the Cactus testing framework http://jakarta.apache.org/cactus to test Struts classes in the actual sever container, it is called in-container testing. It use the actual development environment for testing. To use the CactusStrutsCase you have to extend your test classes from the CactusStrutsCase.
Note: You can change the approaches by simply subclassing the MockStrutsTestCase or the CactusStrustTestCase without changing another line of your code.

The Example

We use an existing example, a library application, to write some tests for. You can download the

full source code of the library application with the link below.

http://www.laliluna.de/assets/tutorials/junit-testing-struts-source.zip

Notice for non MyEclipse User:

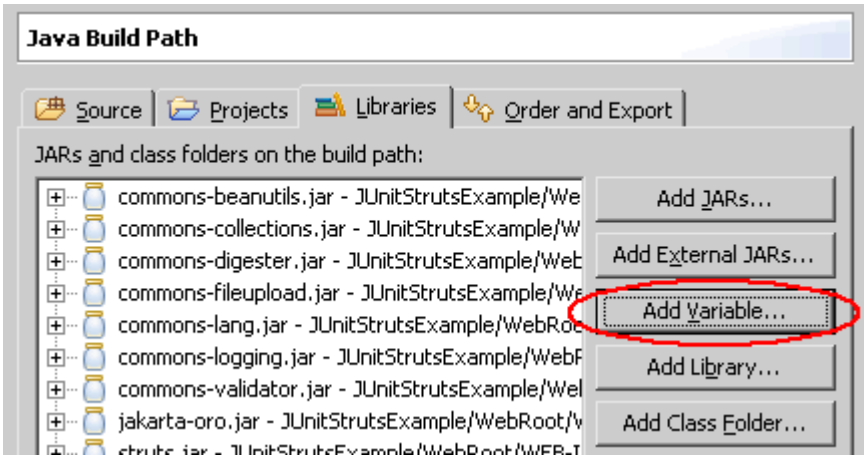
Have a look at the tutorial http://laliluna.de/first-steps-with-struts-free-tools-en.html to learn how to create a Webproject with free tools.

In the example application you can add, edit, delete and list books of a library. We want to test some of this features to make sure that they working correctly.

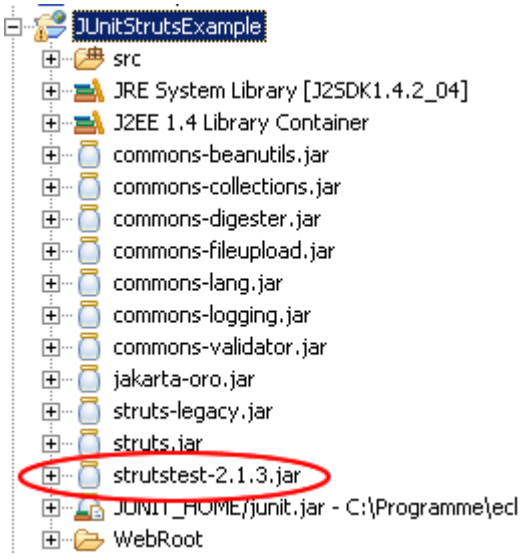
The TestCase

If you have downloaded the example application and import it in the Eclipse Workspace first you have to add the JUNIT_HOME variable to the Java-Build-Path of our project.

Open the properties dialog of the project, right mouse button on the project and choose *Properties*. Open the Java-Build-Path section and click on the *Add Variable..* button on the right side. Choose the JUNIT_HOME variable of the list.



The next step is fit up the application with the *StrutsTestCases* jar file. Copy the jar file into the *lib* folder under */WebRoot/WEB-INF* and refresh the project tree in the package explorer (Download StrutsTestCases).



Let's start by creating an empty test case, which we extend from the base StrutsTestCase class. First, add a new package *test.de.laliluna.tutorial.library*

Create a new class *TestLibrary* which extends from the *MockStrutsTestCase* class of the *StrutsTestCases* extension.

The class is used as a class which extends the TestCase class of the JUnit framework. So you can override the *setUp()* and *tearDown()* method.

Note: If you override the *setUp()* method you must explicit call *super.setUp()*. This method performs some important initializations routines, and *StrutsTestCases* will not work if it is not called.

The first thing we do is create a constructor with a parameter *testName* which calls the super class.

```
public class TestLibrary extends MockStrutsTestCase {

    public TestLibrary(String testName) {
        super(testName);
    }
}
```

We want to write test for the following features of the application:

- Save a book
- List all books

First of all we have initialize some parameters and an action form.

Provide a private property *bookEditForm* of type *BookEditForm*.

```
public class TestLibrary extends MockStrutsTestCase {

    private BookEditForm bookEditForm;

    public TestLibrary(String testName) {
        super(testName);
    }
}
```

Override the *setUp()* method to initialize some parameter.

```
public void setUp() throws Exception {
        super.setUp();

        // set the context directory to /WebRoot
        // to find the /WEB-INF/web.xml
        setContextDirectory(new File("WebRoot"));

        // set the book into the action form
        bookEditForm = new BookEditForm();
        bookEditForm.setBook(new Book(1, "laliluna", "StrutsTestCases Tutorial",
true));

}
```

Use the *setContextDirectory(..)* method to set the context directory where StrutsTestCases search for the web.xml file, because we use MyEclipse and the web.xml is placed in the */WebRoot/WEB-INF/* folder. (Default is WEB-INF/web.xml)

Then create a new instance of the action form class *BookEditForm* and set the property book.

Test save a book

First have a look in the *BookEditAction* class where you find the method *saveBook*. You will see that a action form is used to get the data of the book which will be saved. After saving the book we will find and return a action forward *showList*.

The following listing shows the method *saveBook*:

```
public ActionForward saveBook(
            ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) {
            BookEditForm bookEditForm = (BookEditForm) form;

            /* lalinuna.de 04.11.2004
             * init SimulateDB class and get data by id
             */
            SimulateDB simulateDB = new SimulateDB();
            simulateDB.saveToDB(bookEditForm.getBook(), request.getSession());

            return mapping.findForward("showList");

}
```

Create a new method *testSaveBook* in the class *TestLibrarty* to test if you can add a book without errors.

With the method *setRequestPathInfo* you set the request path for the test case.

We need a parameter *do* which identifies the method which is called by the *DispatchAction* class *BookEditAction*. Use the method *addRequestParameter* to set a parameter and the value.

Within the saveBook method a action form stores the book which will be saved. We have to set this action form with the method setActionForm.

Finally we need to get the Action to do its thing. Execute the *actionPerform* method.

```
public void testSaveBook() {

        setRequestPathInfo("/bookEdit");
        addRequestParameter("do", "saveBook");
        setActionForm(bookEditForm);
        actionPerform();
}
```

That's all you have to do to get the ActionServlet to process your request.

Now we still need to verify that everything happened as we expected it to. First we want to test if the book is added to the collection which is placed in the session. Get the list of books from the session as an ArrayList. The last entry of the list must be the book we have added before, so we test the title of the last entry if it is equal to our title.

Then we verify the action forward which is returned by the *saveBook* method of the *EditBookAction* class.

```
public void testSaveBook() {

        setRequestPathInfo("/bookEdit");
        addRequestParameter("do", "saveBook");
        setActionForm(bookEditForm);
        actionPerform();
        ArrayList bookDB = (ArrayList) getSession().getAttribute("bookDB");
        assertEquals("StrutsTestCases Tutorial", ((Book) bookDB.get(bookDB.size
() - 1)).getTitle());
        verifyForward("showList");
}
```

In the second scenario we want to test if the *BookListAction* class works correctly.

First have a look to the *execute* method of the *BookListAction* class:

```
public ActionForward execute(
            ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) {
            BookListForm bookListForm = (BookListForm) form;


            /* lalinuna.de 03.11.2004
             * init SimulateDB class and set some dummy data
             */
            SimulateDB simulateDB = new SimulateDB();
          bookListForm.setBooks(simulateDB.getAllBooks(request.getSession()));


          return mapping.findForward("showList");
}
```

Create a new method *testBookList*.

First we have to set the request path to /bookList.

We do not need any other parameter so we can process the request with *actionPerform*.

Next we test if the list of books stored in the action form is not null and then we verify the action forward returned by the method *execute* of the *BookListAction* class.

```
public void testBookList(){

        setRequestPathInfo("/bookList");
        actionPerform();
        assertNotNull(((BookListForm)getActionForm()).getBooks());
        verifyForward("showList");

}
```

Run the test cases

Finally we can run the test cases. Right mouse button on the test class TestLibrary and choose *Run* > *JUnit Test*. If all test runs without errors your JUnit tab of MyEclipse looks like the following.