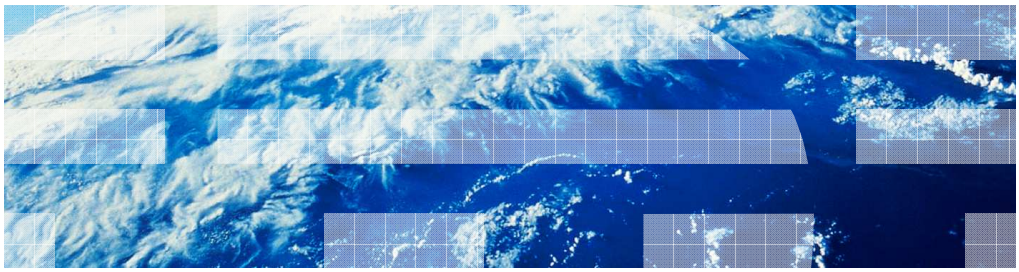
WebSphere Commerce V7 Feature Pack 1

Storefront automation harness



WebSphere software

This presentation provides an introduction to the Storefront automation harness in WebSphere® Commerce V7 Feature pack 1.

Table of content

- Storefront automation harness overview
- Storefront automation harness key assets and their organization
- Review sample codes to understand how a test script is implemented as a JUnit test case

This presentation begins with an overview of Storefront automation harness. Then you will see an introduction of Storefront automation harness key assets and their organization. Finally, the presentation ends with a detailed look at some sample code to help you understand how a test script is implemented as a JUnit test case.

Storefront automation harness

- Software to automate storefront functional verification test
- Use JUnit to run the test scripts
- Use the Selenium tool to perform actions on browsers
- Packages provided
  - Storefront Test Automation Engine
  - Madisons and Elite starter store test bucket projects

Storefront automation harness is a software which allows you to automate the storefront functional verification test. Storefront automation harness uses JUnit to run the test scripts and uses the Selenium tool to perform the actions on browsers. Three projects are included in the storefront automation harness package, including Storefront Test Automation Engine, Madisons and Elite starter store test bucket projects.

How the Storefront automation harness lowers your total cost of implementation

- Do not need to create by your own from scratch
- Use test harness to create your own store's test bucket
- Catch regression defects earlier
- Quickly find the defects during the development phase
- Ease the testing against different browsers

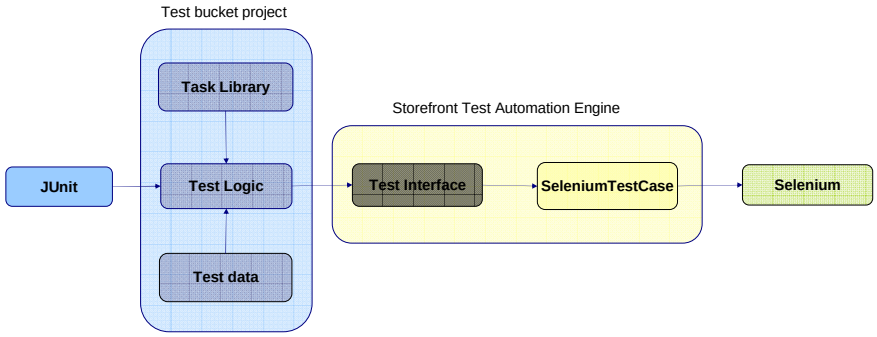
The Storefront automation harness focuses on reducing the test effort associated with storefront functional verification test.

With the Storefront automation harness, you do not have to create your own test project from scratch. You can extend the storefront test bucket provided in the Storefront automation harness package if your store is extended from the Madisons starter store or Elite start server. If your store is not extended from any of these two starter stores, you are still able to use the Storefront Test Automation Engine provided in the Storefront automation harness package to create your own test bucket.

With the Storefront automation harness, you can run test bucket to catch your regression defects earlier when you have a new release. You can catch the defects when they come up during the development phase. You can easily run your storefront functional verification tests on the supported browsers.

Key assets for Storefront automation harness

- Storefront automation harness consists of following assets
  - JUnit – Open source
  - Test bucket projects – Provided in Storefront automation harness
  - Storefront Test Automation Engine – Provided in Storefront automation harness
  - Selenium test tool – Open source



5 Storefront automation harness

Storefront automation harness consists of four key assets: JUnit, test bucket project, Storefront Test Automation Engine and Selenium.

JUnit is a Java™ tool used to run test cases. It is used by the storefront automation harness to initiate tests.

The test bucket project is a Java project which contains test logic, test data and task library packages.

The test logic package contains test scripts. A test script is a Java class which contains all the logic for a test scenario.

The test data package contains data files which hold the actual data for each test script.

The task library package contains the reusable tasks. A library is a Java class. A task is a Java method which implements a storefront action, such as clicking on the header link.

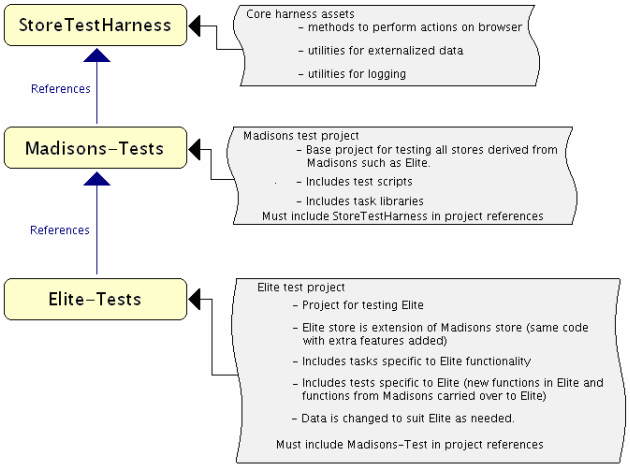
Storefront Test Automation Engine is a Java project which contains the code to interact with Selenium test tool to do the test. Two key packages are included in this project: Test interface and Selenium Test case.

Test interface provides a set of methods, such as click, select, and type. The test scripts use these methods to instruct the Selenium test tool to perform the corresponding action in the actual browser.

Selenium Test case is the implementation of test interface for Selenium tool.

Projects included in the Storefront automation harness package

- Storefront Test Automation Engine
  - StoreTestHarness
- Test bucket projects
  - Madisons-Tests
    - Depends on StoreTestHarness
  - Elite-Tests
    - Extends Madisons-Tests



6 Storefront automation harness

The test assets are organized into two types of Java projects.

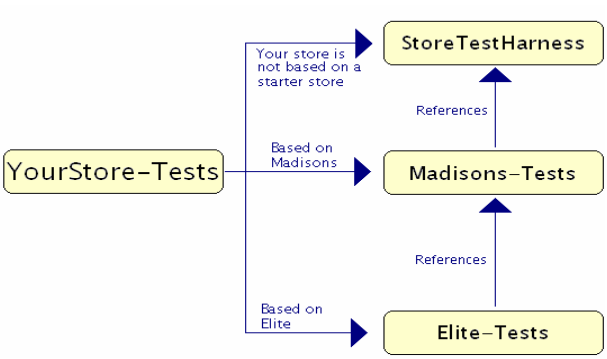
Storefront Test Automation Engine is a Java project which contains the code to actually perform actions in the browser.

Test buckets projects are Java projects which contain all the test scripts and task libraries for a particular store.

Test buckets projects are organized in the same way as the actual stores. Elite store is an extension of the Madisons store. They share the same code base but the Elite store has additional features on top of what Madisons has. The same is true for the Madisons-Tests and Elite-Tests projects. Madisons-Tests contains all the tests for Madisons, while Elite-Tests contains all the same tests that were in Madisons base PLUS tests for Elite specific functionality.

Create your own test bucket project

- Create your own test bucket project instead of updating Madisons or Elite starter store test bucket
  - Create a new test bucket project and reference to StoreTestHarness if your store is not based on Madisons or Elite starter store
  - Create a new test bucket project and reference to Madisons-Tests if your store is based on Madisons starter store
  - Create a new test bucket project and reference to Elite-Tests if your store is based on Elite starter store



7 Storefront automation harness

For your own storefront function verification test, you should not update or change directly the test bucket projects provided in the Storefront test harness. Instead you should create your own test bucket project according to following three scenarios:

If your store is not based on Madisons or Elite starter store, you should create a new test bucket and reference to the project StoreTestHarness.

If your store is based on Madisons starter store, you should create a new test bucket project and reference to project Madisons-Tests.

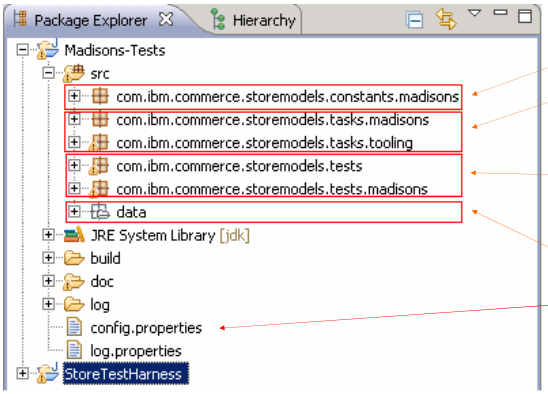
If You store is based on Elite starter store, you should create a new test bucket project and reference to project Elite-Tests.

Storefront Test Automation Engine Dependencies

- One of Eclipse-based software is installed
  - WebSphere Commerce Developer
  - Rational® Application Developer
  - Eclipse
- At least one of the browsers is installed
  - Internet Explorer 7.0 or 8.0
  - Firefox 3.0
- JUnit (Tested with v3.8.2)
- Selenium RC (Tested with v1.0.1)
- Apache HttpComponents Client HttpClient.zip (tested with V4.0.1)

To install the test automation harness, you must have all the software listed here. A link to the WebSphere Commerce Information Center is provided in the Preference slide. You should refer to that link for the complete required software.

Madisons test bucket project



- Five types of asset
  - Constants
  - Task libraries
    - Madisons
    - Tools - setup or clean up test environment
  - Test cases
    - Tests – JUnit test suite to run all tests
    - Madisons – test scripts
  - Data files
  - Properties files

9 Storefront automation harness

The test bucket consists of five key assets: constant, task libraries, test cases, data files and property files.

Constants contains a set of constant files used by the tasks and test scripts.

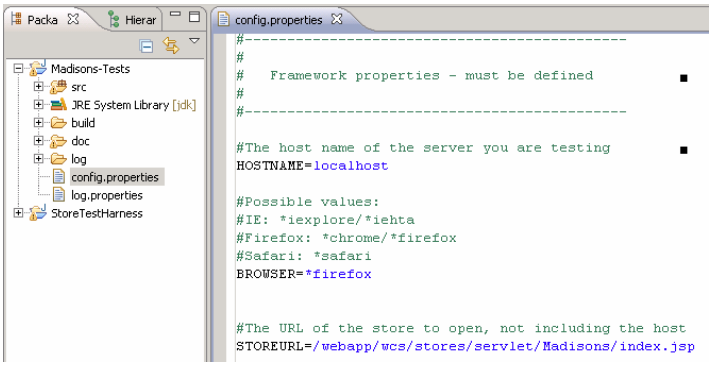
The task libraries contain two packages. The Madisons package contains the tasks for Madisons testing. The tool package contains the tasks for WebSphere Commerce tools to setup or clean up the test environment.

The test cases asset also contains two packages. Tests package contains a JUnit test suite to run all test scripts in the Madisons test bucket. The Madisons package contains the test scripts for the Madisons starter store.

The data files asset contains the test data used by test scripts.

The property files define some global environment variables for the WebSphere Commerce server and for the test client environment.

Global environment variables

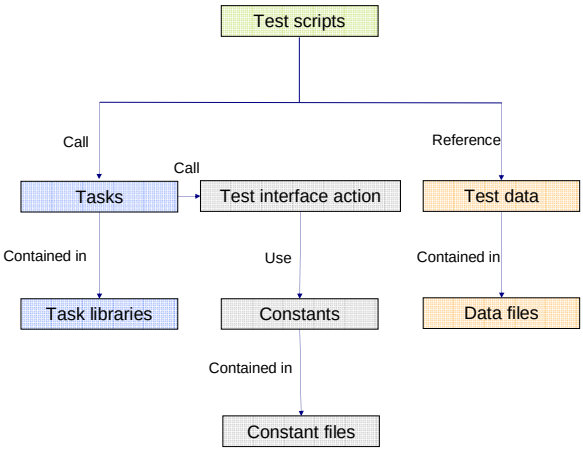


- Check this file first before you run any test
- The config.properties file defines several environment variables. For example:
  - HOSTNAME
  - BROWSER
  - STOREURL
  - DEFAULTTIMEOUT
  - TESTTIMEOUT
  - ADMIN_USER_NAME
  - ADMIN_PASSWORD

In the test bucket project, the config.properties file defines several environment variables which describes your test environment. Before you start to run your test, you should make sure the variables defined here are appropriate for your environment.

Test script

- A Java class that automates a single test scenario
- Contains one or more test cases. Each test case is a JUnit test case method
- Test cases perform storefront actions by calling the tasks in a specific order
- Tasks call the test interface action methods to send action to browser
- Test interface actions use constants to access the storefront objects
- Each test script reference one test data file's data



A test script is a JUnit Java class that automates a single test scenario. Each test script contains one or more test cases. A test case is a Junit test case method which holds the logic to perform storefront actions by combining tasks from the task libraries in a specific order. The tasks are contained in the task libraries. The tasks use the test interface reference to call the test interface action methods to send actions to browser. Test interface uses storefront object ID to access a store's object. The object ID is assigned to a constant and contained in the constant files. The data used for a test case is from a data file. You should have one data file for one test script.

A sample test script

```
public class StoreModelTestCase extends TestCase {
```

junit.framework.TestCase

```
/** Test scenario to test various use cases associated with user registration
 *  Refer to each test case for a detailed use case description
 */
public class FSTOREB2C_01 extends StoreModelTestCase {

    //A variable to hold the name of the data file where input parameters can be found.
    protected final String dataFileName = "data/FSTOREB2C_01_Data.xml";

    //A variable to hold the name of the data file where parameters to clean the store from previous scenarios is located.
    protected final String initDataFileName = "data/CleanStore.xml";

    /**
     * Determine if the store cleanup step has been completed for this scenario yet.
     */
    private boolean initComplete = false;

    /**
     * Perform setup operations such as initializing the test harness and setting the data file to use.
     */
    public void setUp() throws Exception {

        //Starts the test server.
        test.doSetUp();

        if (!initComplete)
        {
            Store.cleanUpStore(initDataFileName);
            initComplete = true;
        }

        //Set the data file to use.
        setDataFile(dataFileName);
    }
```

12 Storefront automation harness

A sample test script is shown here.

This test script extends the StoreModelTestCase class. It in turn extends the JUnit TestCase class. So you can see that a test script is a JUnit test case.

The setUp method is a JUnit method. This method is called before any test case method in this class. In this method, the test interface reference “test” is used to call doSetup method to perform Selenium test tool setup steps. The method setDataFile is called to set the data file used by this test script.

A sample test case in a test script

JUnit 3 use **test** prefix for test case method

Test case block

Data block

```
public void testFSTOREB2C_0101() throws Exception {

    //Tell which test case to use for input parameters
    setDataLocation("testFSTOREB2C_0101", "testFSTOREB2C_0101_1");

    //open store named Madisons using firefox as the browser
    Store.openStore();

    //click on the Register button and fill up the registration fo
    RegistrationPage.registerNewUser(getInputParameter("LOGONID")+

    //verify that user has been registered successfully
    assertEquals (Store.isUserRegistered(), true);
}
```

Use case task

```
public static void registerNewUser(String logonID,
        String userZipCode, String userEmail, Stri
        String userGender, String birthDate, Strin

    Header.clickSignIn();

    RegistrationPage.clickRegister();

    RegistrationPage.verifyIsRegistrationPage();

    RegistrationPage.enterFieldLogonId(logonID);

    RegistrationPage.enterFieldPassword(password);

    //update Password Verify field
    RegistrationPage.enterFieldPasswordVerify(pass

    //update Last Name field
    RegistrationPage.enterFieldFirstName(firstName

    //update Last Name field
    RegistrationPage.enterFieldLastName(lastName);

    //update Street Address field
    RegistrationPage.enterFieldStreetAddress(stree

    //update City field
    RegistrationPage.enterFieldCity(userCity);
```

Store task

13 Storefront automation harness

A sample test case method is shown here.

This test case is to register a shopper from the store home page. In the method, setDataLocation determines which test case block and data block in the data file is used as input data.

The Store task library's openStore task is called to open the store in the browser and the RegistrationPage task library's registerNewUser task is called to register a new user.

The method registerNewUser is a use case task, which consists of multiple store tasks. In the screen capture you can see that RegisterNewUser calls other tasks such as ClickSignIn, ClickRegister, and so on.

This sample code uses JUnit 3.0, so the method has prefix "test". The Storefront Test Automation Engine does not care which version of JUnit you use. If you use JUnit 4, you should use @Test annotation for your test case method.

Data files

- One scenario level data file for one test script
- One data file can have one or more test case block. One test case data block is for one test case
- One test case block can have multiple data block

Test case block

Data block

```
<Scenario name="FSTOREB2C_01">
    <Test name="testFSTOREB2C_0101">
        <Datablock name="testFSTOREB2C_0101_1">
            <Input>
                <Parameter name="LOGONID" value="shawn"/>
                <Parameter name="FIRST_NAME" value="John"/>
                <Parameter name="LAST_NAME" value="Taverez"/>
                <Parameter name="PASSWORD" value="Shopper1101"/>
                <Parameter name="PASSWORD_VERIFY" value="Shopper1101"/>
                <Parameter name="ADDRESS" value="8200 Warden Ave"/>
                <Parameter name="CITY" value="Markham"/>
                <Parameter name="STATE" value="Ontario"/>
                <Parameter name="COUNTRY" value="Canada"/>
                <Parameter name="ZIPCODE" value="L3G 1H2"/>
                <Parameter name="EMAIL" value="john@test.test"/>
                <Parameter name="PHONE_NUMBER" value="905-413-2000"/>
                <Parameter name="SEND_UPDATES" value= "NULL"/>
                <Parameter name="PREFERRED_LANGUAGE" value="United Stat
                <Parameter name="PREFERRED_CURRENCY" value="US Dollar"/
                <Parameter name="GENDER" value="Male"/>
                <Parameter name="BIRTH_DATE" value="1"/>
                <Parameter name="BIRTH_MONTH" value="3"/>
                <Parameter name="BIRTH_YEAR" value="1980"/>
                <Parameter name="AGE" value= "NULL"/>
                <Parameter name="MOBILE_COUNTRY" value = "Canada"/>
                <Parameter name="MOBILE_PHONE" value = "905-413-0001"/>
                <Parameter name="SEND_MOBILE_NOTIFICATION" value="NULL",
                <Parameter name="SEND_MOBILE_PROMOTIONS" value="NULL"/>
                <Parameter name="REMEMBER_ME" value="true"/>
                <Parameter name="PRIVACY_POLICY_ALERT" value="You are u
            </Input>
            <Output/>
        </Datablock>
```

14 Storefront automation harness

The data file used by registering a new user test case is shown here.

A data file is used by one test script. A data file can have several test case block data. Each test case can have multiple data blocks.

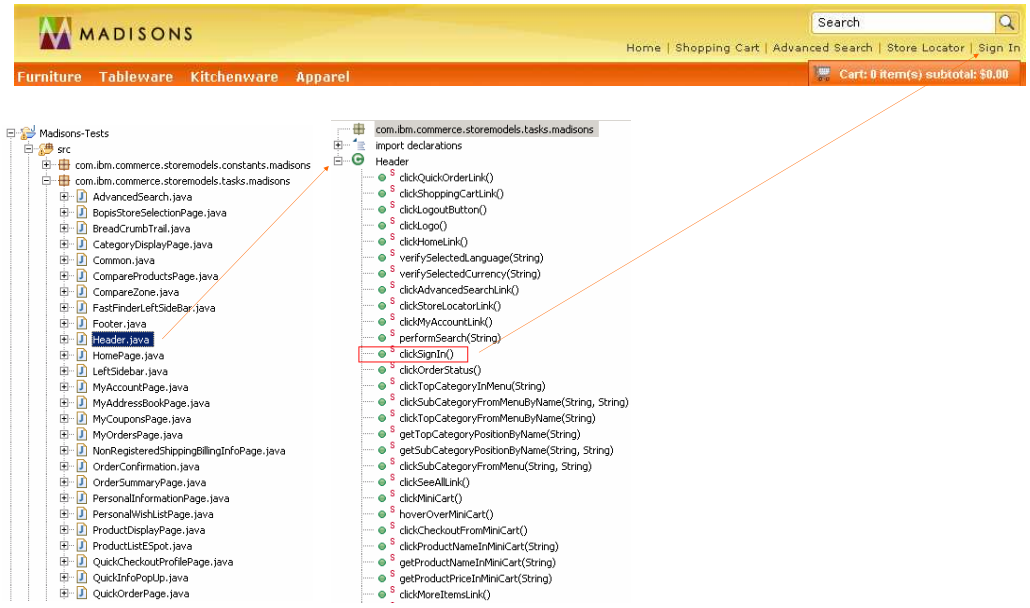
Task library

- One page or page fragment per library
- contains all the tasks that can be performed on that store page or page fragment
- A task is a Java method that interacts with a specific storefront object

A task library in the storefront automation harness consists of all the tasks that can be used to perform actions in the web browser. The task libraries are organized in a way such that one library is for one page or page fragment.

A task in the task library is a Java method that interacts with a specific element on the storefront.

An example of task library



16 Storefront automation harness © 2010 IBM Corporation

As an example, the Header page fragment library is shown here. This library implements all the tasks that can be performed in the header. For example, clickSignIn task implements the ‘click’ action of the Sign In link.

Task method

- Java method that interacts with a specific object on the storefront
- Call test interface reference “test” and action method
- In action method, the store object **constant** is used to reference the object ID
  - Object ID is an HTML tag element ID
    - For example: <a id=“headerLogin”>
  - IDs must be unique in the store pages

A test method is a Java method that interacts with a specific object on the storefront. To perform actions on the storefront pages, a single instance of test interface is used to call the test interface action methods.

In the test interface action methods, the constants are used to reference storefront object IDs. An object ID is an HTML element ID and all the store object IDs are stored in the StoreConstants class. For your own custom store page, you must make sure the IDs are unique.

A sample task method– clickSignIn

- ClickSignIn task is for click action on **Sign In** link in the header
- Constant HEADER_SIGN_IN_LINK reference **headerLogin** object ID

Header.java

```
/**
 * Clicks the Sign In link in the header.
 *
 * Pre-requisites: <ul>
 *                 <li>The header is visible and no user has logged in</li>
 *               </ul>
 * Post conditions: The user is sent to the Sign In page.
 * Flex flow options: none.
 */
public static void clickSignIn(){
    Header.verifyIsHeader();
    test.waitForElement(StoreConstants.HEADER_SIGN_IN_LINK);
    test.click(StoreConstants.HEADER_SIGN_IN_LINK);
}
```

- Madisons-Tests
  - src
    - com.ibm.commerce.storemodels.constants.madisons
      - AcceleratorConstants.java
      - ChangeFlowOptions.java
      - ChangeFlowOptionsUI.java
      - CMCConstants.java
      - ScreenshotConstants.java
      - StoreConstants.java
    - com.ibm.commerce.storemodels.tasks.madisons

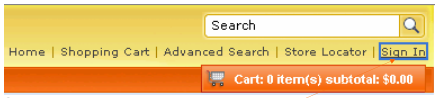
```
/**The Id of Store Locator link found in store header*/
public static final String HEADER_STORE_LOCATOR_LINK = "WC_Cache
/**The Id of search term text field found in store header (simpl
public static final String HEADER_SEARCH_TERM_INPUT_FIELD = "Sim
/**The Id of search button (magnifying glass image) found in sto
public static final String HEADER_SEARCH_BUTTON = "WC_CachedHead
/**The Id of Sign In link found in store header given no user is
public static final String HEADER_SIGN_IN_LINK = "headerLogin";
/**The Id of Order Status link found in store header given no us
public static final String HEADER_ORDER_STATUS_LINK = "WC_Cached
```

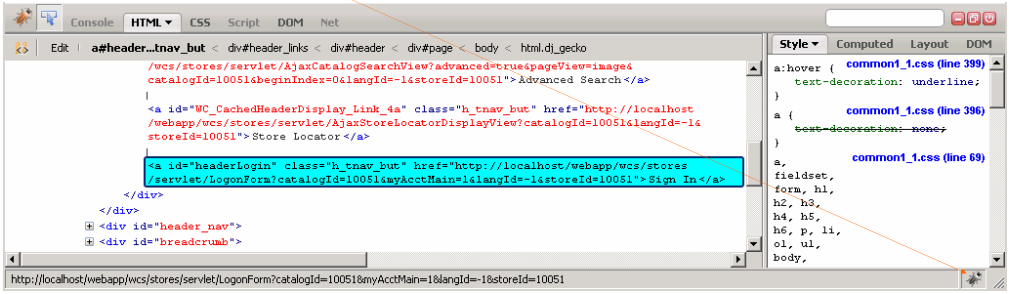

A sample task method clickSignIn is shown here.

The task will click the Sign In link in the header. It calls the click method with test interface reference “test”. The click method uses HEADER_SIGN_IN_LINK constant to retrieve the headerLogin object ID, which is the Sign In anchor ID in the store header.

Find an object ID

- Pre-requisite
  - All objects IDS must be unique in stores
  - Firebug add-on for Firefox is installed
- Sample: Finding the ID of the **Sign in** link in Madisons starter store header
  - Launch Madisons starter store home page in Firefox
  - Click the **Firebug** add-on icon in Firefox status bar
  - mage of the blue rectangle with a cursor over it
  - Hover you ver the **Sign In** link in the header an

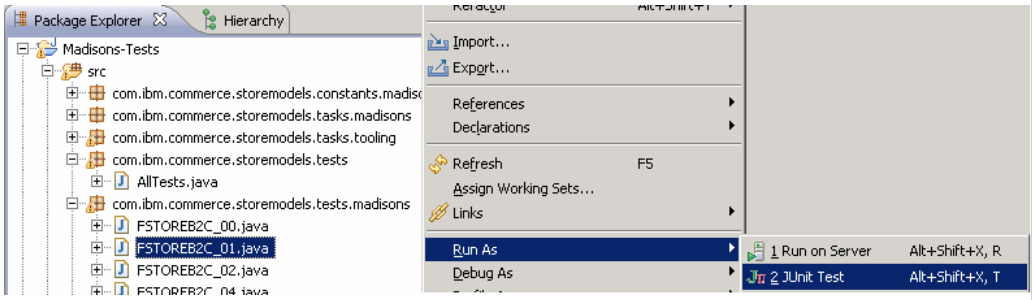




You can find an object ID with Firefox add-on Firebugs by following the steps shown above.

To run a test script

- Test script **FSTOREB2C_00** should be run first to ensure reusable data is setup
  - For example: set up user name and password
- To run a **single test** script
  - Right click test script Java class name > Run As >JUnit Test

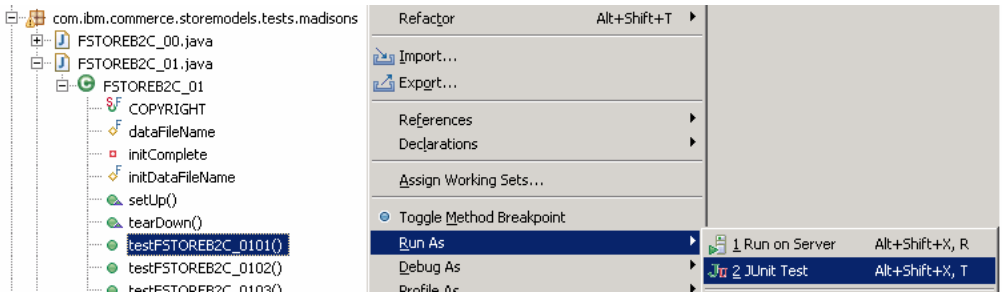


Before you can run any tests in the bucket, you must run the FSTOREB2C_00 test script to set up the test environment, such as set up user name and password.

To run a single test script, right click the test script Java class, select Run As and then JUnit Test.

To run a signal test case

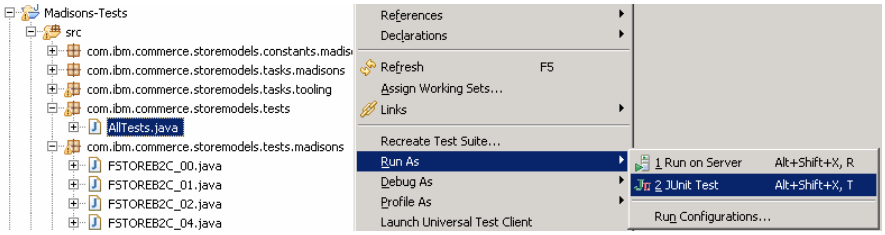
- Expand test script Java class
- Right click the test case method > Run As > JUnit Test



To run a signal test case, expand the test script Java class, right click a test case method, select Run As and then JUnit Test.

To run all test script or run test scripts in batch mode

- To run **all test** scripts in the test bucket
  - Right click AllTests.java > Run As > JUnit Test
- To run all test scripts or several test scripts in batch mode
  - Refer to Information Center for details



To run all test scripts, right click the AllTests.java, as shown in the screen capture, run it as a JUnit test.

You can also run all the scripts or several scripts in the test bucket in batch mode. You can refer to the WebSphere Commerce Information Center for the details.

Summary

- Introduced storefront test harness
- Introduced the test script, task library, object ID constant
- Covered how to run the test scripts

In this presentation, storefront test harness, test script, task library and object ID constant have been introduced. The different ways to run the test scripts are also covered.

Reference

- Selenium-RC

  http://seleniumhq.org/docs/05_selenium_rc.html
- Junit

  http://www.junit.org/
- HttpComponent Client

  http://hc.apache.org/
- Storefront Development and Test Assets

  https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?lang=en_US&source=swg-stae
- Storefront Test Automation Engine dependencies

  http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.madisons-starterstore.doc/tasks/tsminstallharnessinrad.htm

This slide contains some useful references for helping you understand the material.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_FVTTestHarness.ppt

This module is also available in PDF format at: ../FVTTestHarness.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information