

# Tutorial

Here is a brief description of how you can use HttpUnit with JUnit to test your own web sites.

## Obtaining a Web Page Response

- The center of HttpUnit is the WebConversation class, which takes the place of a browser talking to a single site.
- It is responsible for maintaining session context, which it does via cookies returned by the server.
- To use it, one must create a request and ask the WebConversation for a response.

For example:

```
WebConversation wc = new WebConversation();
WebRequest req = new GetMethodWebRequest(
"http://www.meterware.com/testpage.html" );
WebResponse resp = wc.getResponse( req );
```

- The response may now be manipulated either as pure text (via the getText() method), as a DOM (via the getDOM() method), or by using the various other methods described below.
- Because the above sequence is so common, it can be abbreviated to:

```
WebConversation wc = new WebConversation();
WebResponse resp = wc.getResponse(
"http://www.meterware.com/testpage.html" );
```

## Examining and Following Links

- The simplest and most common form of navigation among web pages is via links. HttpUnit allows users to find links by the text within them, and to use those links as new page requests.
- For example, the following page contains a link to the JavaDoc for the WebResponse class, above. That page could therefore be obtained as follows:

```
WebConversation wc = new WebConversation();
WebResponse resp = wc.getResponse(
"http://www.httpunit.org/doc/cookbook.html" ); // read this
page
WebLink link = resp.getLinkWith( "response" );
// find the link
```

```
link.click();
// follow it
WebResponse jdoc = wc.getCurrentPage();
// retrieve the referenced page
```

- Image links can be found as well.
- The `WebResponse.getLinkWithImageText()` method can look up links by examining the ALT text, or the `httpUnitOptions.setImagesTreatedAsAltText` method can cause ALT text to be treated as ordinary searchable text.

### Using the Table Structure of a Web Page

- Many web designers make heavy use of tables to control the page formatting. You can take advantage of this by looking at the tables in the page as discrete elements.
- The `:getTables()` method will return an array of the top-level tables in the page (that is, those which are not nested within other tables), in the order in which they appear in the document.
- Given a table, you can ask for one of its cells, and treat the result either as text or a DOM or ask for and tables, links, or forms nested within it.
- For example, the following code will confirm that the first table in the page has 4 rows and 3 columns, and that there is a single link in the last cell of the first row:

```
WebTable table = resp.getTables()[0];
assertEquals( "rows", 4, table.getRowCount() );
assertEquals( "columns", 3, table.getColumnCount() );
assertEquals( "links", 1, table.getTableCell( 0, 2
).getLinks().length );
```

- In most cases, the purpose of a test is to verify the text, rather than the formatting. `HttpUnit` provides shortcuts to do this. For example, imagine that the second table in the page should look like this:

| Name  | Color |
|-------|-------|
| gules | red   |
| sable | black |

- The following JUnit code will verify it:

```
String[][] colors = resp.getTables()[1].asText();
assertEquals( "Name", colors[0][0] );
assertEquals( "Color", colors[0][1] );
assertEquals( "gules", colors[1][0] );
assertEquals( "red", colors[1][1] );
assertEquals( "sable", colors[2][0] );
assertEquals( "black", colors[2][1] );
```

- The `asText()` method, used to convert an entire table into an array of `String` ignores all formatting tags. At times, entire rows or columns of a table may be used for appearance purposes only.
- You can ignore rows or columns consisting only of images or empty tags by invoking `purgeEmptyCells` on the table.
- In many cases, tables begin with some fixed text and the purpose of the test is to verify the dynamic portion.
- As a general rule, the first cell will contain something fixed and you can search for it - even if the table is nested inside other tables. Thus, the above table could have been found with:

```
String[][] colors = resp.getTableStartingWith( "Name" );
```

- This has the advantage of being immune to many page formatting changes which might make it the third or fourth table.

## Working with Forms

- A dynamic web site tends to have many html forms, each of which contains various kinds of controls: text boxes, pull-down menus, radio buttons, and so on.
- The HTML for these controls varies widely; however, the intent is roughly the same, so `HttpUnit` makes them look the same.
- There are a few basic things that a tester is likely to want to do with a form.
- The most obvious first step is to verify the controls and their default values; if they are correct, the tester will generally make some changes and submit the form to see how the system responds.
- The examples below assume the following form definition:

Restaurant Name:

Restaurant Type:  Chinese  Tex/Mex  Italian

Location:

Accepts Credit Cards

- The default values may be checked as follows:

```
WebForm form = resp.getForms()[0]; // select the first form
in page
    assertEquals( "La Cerentolla", form.getParameterValue(
"Name" ) );
    assertEquals( "Chinese",      form.getParameterValue(
"Food" ) );
    assertEquals( "Manayunk",     form.getParameterValue(
"Location" ) );
    assertEquals( "on",          form.getParameterValue(
"CreditCard" ) );
```

- Note that all controls are treated alike, with the exception of the checkbox.
- Simulating the submission of the form can be done most simply by obtaining the form object and calling its 'submit' method, possibly modifying the form parameters beforehand.
- For example, to correct the restaurant type and indicate that it does not accept credit cards:

```
    form.setParameter( "Food", "Italian" );           // select
one of the permitted values for food
    form.removeParameter( "CreditCard" );           // clear
the check box
    form.submit();                                   // submit
the form
```

And of course the test could then proceed to examine the response to this submission as well, obtaining it from `wc.getCurrentPage()` .

## Working with frames

Without frames, web interaction tends to be straightforward and sequential. There is one current active page at a time, and each new page replaces the old one that referenced it. Frames change that, allowing multiple active pages simultaneously, and allowing for the possibility that a link from one active page could result in the replacement of a different page.

- HttpUnit supports frames by providing methods on the `WebConversation` class to examine those frames which are currently active.

- Each response replaces the contents of the appropriate frame, which is not necessarily the topmost frame ("\_top"). I
- In the following scenario, there are two active subframes, named "overview" and "details":

```
WebConversation wc = new WebConversation();
    WebResponse top      = wc.getResponse(
"http://www.meterware.com/Frames.html" ); // read a page
with two frames
    WebResponse summary = wc.getFrameContents( "summary" );
// read the summary frame
    WebLink      link    = summary.getLinkWith( "Cake
Recipe" );              // find the link (which
targets "details" );
    link.click();
// click on it
    WebResponse response= wc.getFrameContents( "details" );
// retrieve the details frame
```