DocV1.WebUnittests

Web Unit tests

When a Java developer gets used to unit tests it wants to test everything. But testing user web interfaces is not that easy. It certainly has nothing in common with basic services unit tests. One would like to create test suites that could simulate user behavior and assert the returned content with the HTTP response. Actually, we have envisioned such a tool since we used the Sun Microsystems TCK test suite, when we confirmed our JSR168 compliance.

To develop such an in container web unit test framework, we used the HTTPUnit library and built a programmatic API on top of it. Hence, it is quite easy to test portlets' web interfaces, and creating a test suite that can simulate a user session can be done very quickly. For example, it is possible to assert that once the link A in portlet block B has been clicked, the returned markup language contains the text T. Many possibilities are available and we are steadily creating more methods to test ever finer conditions.

```
goToPage(path) ; addWebUnit(new ClickLink('ClickEditorLink', 'Try to
open the editor'). setTextLink('Editor'). setBlockId('file-explorer').
addValidator(new ExpectFormNameValidator('contentForm')));
addWebUnit(new SubmitFormUnit('PreviewContent', 'Preview the content').
setFormName('contentForm'). setField('op','preview).
setField('content','Test create new file'). setBlockId('file-explorer').
addValidator(new ExpectLinkWithTextValidator('Back'))) ; addWebUnit(new
ClickLink('GoBackToEditor', 'Go back to the editor').
setTextLink('Back'). setBlockId('file-explorer'). addValidator(new
ExpectFormNameValidator(contentForm))); addWebUnit(new
SubmitFormUnit('SaveContent', 'Save Content with file name empty').
setFormName('contentForm'). setField('op', 'saveContent').
setField('content', 'Test create new file').
setBlockId('file-explorer'). addValidator(new ExpectTextValidator('You
need to enter a name for the file'))) ;
```

We then add these test suites in one class that thereby wraps all the actions a specific user would do. Actually, we provide two classes like that : NormalUser and AdminUser. Obviously, the AdminUser gets more privileges as we simulate a login with the correct login and password. It is thereby possible to test the correct use of rights within the portal and portlets, as well as using the admin user to first configure the states that a normal user would need - for example the creation of portlet categories and portlets in the portlet registry to allow a normal user to add those configured portlets in his pages.

During the last phase of the platform development we decided to create a test suite for all the portal features (mainly the WYSIWYG customizer) and for all portlets. The main advantage is to allow easy refactors of portlets and portlet frameworks. Indeed, a regressive change in the Java Server Faces bridge, or in one of the default UI components, would be directly discovered during the execution of the web test unit suite.

Furthermore, we can validate if the returned markup is xHTML valid, and a CSS parser to test for CSS2 compliance is also being integrated.

We use this web test framework as a stress test tool as it is possible to configure the execution of several threads. This way we simulate various client types' simultaneous interactions with the portal. The only difference with the previous use (ie : web unit test) is that this time the number of threads defined in the test framework's properties file is higher than one. The following test simulates 50 users requesting the portal at the same time.

```
[java] task left: 0 [java] Suite Name: NewAccountSuite [java]
Description: Create a new user account [java] Unit Name Counter Error
Malformed Avg(ms) Avg(kb) Sum(kb) [java] NewSession 250 0 0 1178 21.66
5415.059 [java] GoToMyPortalPage 250 0 0 476 18.277 4569.404 [java]
CreateNewAccount 250 0 0 1351 18.391 4597.864 [java] [java] Suite Name:
LoginSuite [java] Description: Go to the home page and login, using web
client name for user name and passw ord [java] Unit Name Counter Error
Malformed Avg(ms) Avg(kb) Sum(kb) [java] NewSession 250 0 0 970 18.391
4597.864 [java] Login 250 0 0 1140 25.218 6304.556 [java] [java] Suite
Name: RandomPageBrowse [java] Description: Go to the various pages in
exo portal [java] Unit Name Counter Error Malformed Avg(ms) Avg(kb)
Sum(kb) [java] GoToProductPage 250 0 0 645 23.101 5775.434 [java]
GoToServicesPage 250 0 0 893 22.994 5748.718 [java] GoToCompanyPage 250
0 0 798 21.285 5321.4 [java] GoToPressPage 250 0 0 670 21.945 5486.292
[java] GoToCommunityPage 250 0 0 511 25.897 6474.292 [java]
GoToMyPortalPage 250 0 0 376 21.71 5427.696
```

These stress tests highlight the portlets that take too much time to render their markup, and when run for several hours, the existence of memory leaks.

This double purpose is quite powerful as it bypasses the need to configure several tools like JMeter. Consequently, the portlet developer has more time to focus on the creation of a well detailed test suite that is highly reusable.

As a side effect, we have also decided to use this web test framework for producing up-to-date high level documentation for expert users. Indeed, when a portlet developer creates a test suite he basically has to simulate all the use cases. As for any type of unit tests, they always have to be in synchronisation with the portlet features. Therefore, if each use case phase is well described with

comments, it is possible to produce HTML or PDF reports after the test has been executed. By producing XML reports we allow XSLT/XSL manipulation of test data to produce documentation reports. As stated, this documentation is not intended for end users, but for expert users and platform admins who may get up-to-date reports on the portlets they use.

DocV1.WebUnittests (en)
CrÃ©ateur: XWiki.benjamin.mestrallet Creation Date: 2005/02/09 20:34
Dernier Auteur: XWiki.benjamin.mestrallet Last Modification Date: 2005/02/15 22:55