JUnit testing DomUI

Non-visual code can be JUnit tested as usual. But as a visual UI framework testing DomUI itself brings some challenges. While still a work in progress this is what is currently done to help with testing the visual and browser based aspects of DomUI.

Selenium based testing

The visual tests all use Selenium WebDriver to run tests. To run these we actually need two parts operational:

- A web server which runs the several DomUI pages under tests. Components are tested by creating special pages for them that exhibit the behavior we want to test. These pages are all present in the DomUI DEMO application (to.etc.domui.demo).
- A JUnit test runner which then runs a test. The test controls a browser using Selenium WebDriver, and uses Selenium tests written in Java to exercise the pages running on the web server.

The Maven build runs these tests as *Integration Tests*. These can be found automatically (by Maven) because the class names all start with **IT**, like ITHtmlEditorComponentTest. Them being Integration Tests means that they run only after all "normal" tests have run, and after the whole web application has been constructed. This works as follows:

- Maven does the normal build cycles until after *package*. This is the normal compile, and as part of it the per-module JUnit tests are executed as usual.
- After the package phase Maven starts a Jetty server with the to.etc.domui.demo web application deployed on it on port 8088.
- Now Maven locates all Integration Tests and runs them. These tests use localhost:8088 as the base URL and so connect to the Jetty server started by Maven.
- Once done Maven stops the Jetty server and reports the test results.

You can test against any Selenium-supported browser by setting properties inside ~/.test.properties, or passing them as Java properties to the JVM. But by default the tests will run using the Chrome Headless browser. This means that chrome and chromedriver need to be installed on whatever station DomUI is built on, as follows:

- Install Chrome as usual
- Download the latest chromedriver from here.
- Unzip the file. This will give you a single file, "chromedriver".
- Copy this file to some directory on your PATH, like /usr/bin for Linux, or ~/bin if you have no rights.

Using Headless Chrome

Since Chrome 59 we can also use Chrome in headless mode, and as PhantomJS development has been abandoned this is now the default.

There are some issues with using chrome. The most important issue is that ChromeDriver/Chrome does not properly take screenshots. Unlike the other drivers chrome takes only partial screenshots of the page, and that breaks some tests and makes bugs harder to find.

To circumvent this issue the DomUI wrapper around WebDriver has a special implementation of the code that creates a screenshot for Chrome. The code is described on Stackoverflow.

Alternative: using Phantomjs

You can also use PhantomJS as the headless test platform. Just install the "phantomjs" executable in some directory in your PATH, and define webdriver.hub=phantomjs in .test.properties or on the command line (-Dwebdriver.hub=phantomjs).

Sadly enough PhantomJS is no longer supported because the main developer quit 8-/

Please install phantomjs from the website, and do **not** use your distribution's version of it (so do not use apt-get). The distributions often distribute handicapped versions causing odd test failures.

Using HTMLUNIT

I moved to Phantomjs because its alternative, htmlunit, does not allow screenshots to be taken from the pages, and this makes a lot of tests hard to use. But it can still be used, with the effect that some tests will not really work.

Test helper base classes

The DomUI code wraps Selenium in a few helper classes that help with easier testing of DomUI code. See the provided JUnit tests for details. These classes are thin wrappers around WebDriver itself so missing functionality is easily added.

The base class to write Selenium tests is called AbstractWebDriverTest, and it handles everything to access a webdriver wrapper: just call wd() inside any test to get the proper WebDriver wrapper.

Rendering tests

A lot of tests can be written by just querying the DOM state as delivered by DomUI and its javascript during a test. These tests are all quite simple: just click on buttons, enter texts, then check the resulting DOM in the browser.

But there are tests that we cannot do with this. A good example is the test for the HtmlEditor. This component contains a lot of Javascript, and the actual DomUI node (a TextArea) actually gets replaced by an IFRAME by the editor's Javascript. Other tests require that a layout is fully correct, and that is also difficult to do with just DOM matching.

For this we can use Selenium's ability to take screenshots. Using screenshots of a rendered page we can load the screenshot, then use Selenium's knowledge of the *position* and *size* of a web page element to extract from the screenshot the *actual rendering of the component* as a bitmap. This bitmap can then be further analyzed.

Tests and the Travis-CI build

Failed test screenshots

The UI tests that run during the build will make screenshots for those tests that fail. These screenshots are stored inside the [module]/target/failsafe-reports directory as classname_testname.png, for instance:

```
display ./to.etc.domui.demo/target/failsafe-reports/ITTestLookupInput_testInitialLayout.png
```

If the build fails all of the failsafe-reports directories are collected and tarred, and these are copied to the deployment server so that they can be obtained (they are copied to deployer@xxx/reports.tgz).

The version of phantomjs inside the build

Travis-CI has phantomjs default in its image, but that is an older version (2.0.0). To ensure that we have the same results locally and in the build the build script will download version 2.1.1 of phantomjs and cause it to be used by setting the PATH to it.