Creating and Using a Testbundle

In This Topic

The XperienCentral test framework is based on JUnit. See the JUnit website for an introduction to the framework and documentation (for example on the use of "asserts'). A testbundle is an executable Java application that is used to test other parts of the application. A plugin developer can use this framework to create testbundles for the plugins the developer writes. The big advantage of testbundles is that they are very easily incorporated into the build process. The execution of the tests can be automated, which saves a lot of time testing the plugin's functions. It also contributes to higher quality of the developed plugins.

Creating a Testbundle

Creating a blank testbundle is done based on an archetype, just like creating a basic element, panel, etc. The commands below are based on the directory structure as created in Quick Start.

To create a testbundle, follow these steps:

1. Open a command prompt.
2. Navigate to the directory in which you already created your helloworld plugins (`C:\GX\helloWorld` for example).
3. Enter the command (replace `<XperienCentral Version>` with the version number of XperienCentral that you're building against, (10.10.0, for example):

   ```
   mvn archetype:generate -DinteractiveMode=false -DarchetypeGroupId=nl.gx.webmanager.archetypes -
   DarchetypeArtifactId=XperienCentral-testbundle-archetype -DarchetypeVersion=<webmanager version> -
   DgroupId=com.gxXperienCentral.helloworld -DartifactId=helloworldtestbundle -Dclassprefix=HelloWorld -s ..
   \XperienCentral9\settings.xml
   ```

A new folder called "helloworldtestbundle" is created

Back to Top

Building and Deploying the Testbundle

To build and deploy the testbundle, follow these steps:

1. Open a command prompt.
2. Navigate to the folder `C:\GX\helloWorld\helloworldtestbundle`.
3. Execute the command:

   ```
   mvn -s ..\..\XperienCentral\settings.xml clean package
   ```

4. Copy the file `helloworldtestbundle-1.0.0.jar` from the "target" directory to `C:\GX\XperienCentral\work\deploy`.

Back to Top

Running the Testbundle

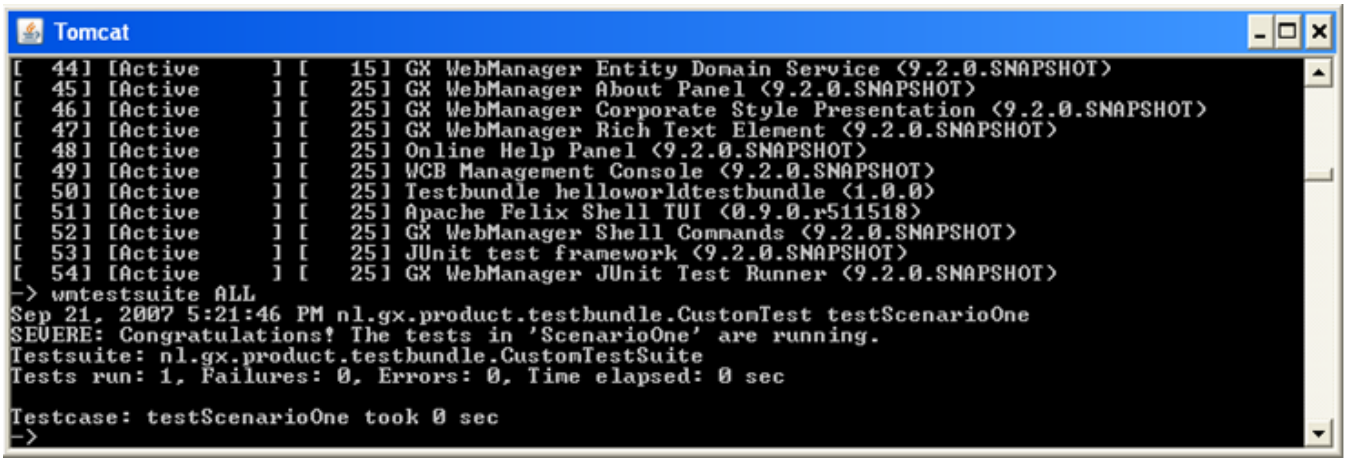
The testbundle is now deployed in the XperienCentral release. The easiest way to run the test it is from the Tomcat console. In order to run this test from the console, a few extra bundles need to be present. Therefore, copy the following bundles to `C:\GX\XperienCentral9\work\deploy`:

- `Webmanager-webapps\Webmanager-backend-webapp\target\Webmanager-backend-webapp-x.x.x\WEB-INF\bundles\org.apache.felix.shell.tui-1.0.0.jar`
- `Webmanager-webapps\Webmanager-backend-webapp\target\Webmanager-backend-webapp-x.x.x \WEB-INF\bundles\XperienCentral-testrunner-bundle-x.x.x.jar`
- `Webmanager-webapps\Webmanager-backend-webapp\target\Webmanager-backend-webapp-x.x.x \WEB-INF\bundles\XperienCentral-junit-bundle-x.x.x.jar`
- `Webmanager-webapps\Webmanager-backend-webapp\target\Webmanager-backend-webapp-x.x.x \WEB-INF\bundle\XperienCentral-felix-shellcommands-x.x.x.jar`

After adding these JAR files to the `/deploy` directory, enter the following command in Tomcat:

```
wmtestsuite ALL
```

The result will be the following lines in the Tomcat log:



Back to Top

Adding Tests to the Default Testbundle

To add tests to the testbundle, only one file needs to be extended: `CustomTest.java` By default, `CustomTest.java` looks like this:

```
public class CustomTest extends TestCase {

        private static final Logger LOG = Logger.getLogger(CustomTest.class.getName());

        protected void setUp() {
    }

        protected void tearDown() {
    }

        public void testScenarioOne() {
                // Place your tests in this method

                LOG.log(Level.SEVERE, "Congratulations! The tests in 'ScenarioOne' are running.");

        }
}
```

The JUnit framework will process the methods in the following order:

1. Run the `setUp` method.

2. Run all methods that start with "test" - the order in which they will be processed is not fixed.
3. After running the tests, the method `tearDown` will be called.

Back to Top

Accessing the XperienCentral API from the Testbundle

To access the XperienCentral APIs, the authorization needs to be set correctly. Typically, the authorization needs to be in place for all the tests that will be run from the testbundle, therefore it is recommended that you make the XperienCentral session object available to all the tests by using private class members. A testbundle that needs XperienCentral authorization will typically look like this:

```
public class CustomTest extends TestCase {

        private static final Logger LOG = Logger.getLogger(CustomTest.class.getName());

        private static final String USERNAME = "Administrator";

    private static final String PASSWORDCLEANBUILD = "Administrator";

    private static final String PASSWORDAFTERLOGIN = "123456";

    private static final String WEBID = "26098";

        private Session mySession = null;

    private ElementManagementService myElementService = null;

    private PageManagementService myPageService = null;

    private MediaRepositoryManagementService myMediaRepositoryService =null;

    protected void setUp() {
                initializeServices();
    }

    protected void tearDown() {
    }

    public void testScenarioOne() {
                // Place your tests in this method

                LOG.log(Level.SEVERE, "Congratulations! The tests in 'ScenarioOne' are running.");

        }

    private void initializeServices() {
                // Contents of this method is presented on the next page
    }

}

private void initializeServices() {
        try {
                LOG.log(Level.INFO, "Retrieving SessionManager from Service Framework");
                   Framework framework = FrameworkFactory.getInstance().getFramework();
                SessionManager sessionManager = null;
                AuthorizationService authorizationService = null;
                ConfigurationManagement configService = null;
                try {
                        sessionManager = (SessionManager)
                        framework.getService(SessionManager.class.getName());
            authorizationService = (AuthorizationService)
                        framework.getService(AuthorizationService.class.getName());
                        configService = (ConfigurationManagement)
```

```
                        framework.getService(ConfigurationManagement.class.getName());
                } catch (FrameworkException e) {
                        LOG.log(Level.SEVERE, "Error in retrieving the SessionManager from the framework.", e);
        }
        String portnr = configService.getParameter("website_settings.frontend_portnr");
        String hostname = configService.getParameter("website_settings.backend_hostname");
                MockServletContext context = new MockServletContext();
                MockHttpServletRequest request = new MockHttpServletRequest(context);
                request.addParameter("webid", WEBID);
                request.setServerName(hostname);
                request.setServerPort(Integer.parseInt(portnr));
                MockHttpServletResponse response = new MockHttpServletResponse();
        mySession = sessionManager.createSession(request, response);
        if(!authorizationService.login(USERNAME, PASSWORDCLEANBUILD, request)) {
                LOG.severe("Login failed with username '" + USERNAME + "' and password '" + PASSWORDCLEANBUILD
+ "'. Another
                        login attempt is made with password '" + PASSWORDAFTERLOGIN + "'.");
            if (!authorizationService.login(USERNAME, PASSWORDAFTERLOGIN, request)) {
                    LOG.severe("Second login attempt also failed.");
            }
                }
                myElementService = mySession.getElementManagementService();
        myPageService = mySession.getPageManagementService();
                myMediaRepositoryService = mySession.getMediaRepositoryManagementService();
        } catch (ConfigurationManagementException e) {
                LOG.severe(e.toString());
        e.printStackTrace();
    }
}
```

Back to Top

Using the Spring Mock Framework

XperienCentral integrates the Spring mock classes to make it more convenient to write JUnit testbundles. For example, this can be used to emulate a request and response. More information about Spring mock can be found at junit.org.

Back to Top