EUSKO JAURLARITZA
GOBIERNO VASCO

Programa Saiotek 2006

SMARTLAB

Entorno de Trabajo Inteligente Colaborativo y Programable

Testing in OSGi

Tecnológico Fundación Deusto
Teknologikoa Deustu Fundazioa

HISTORIAL DE CAMBIOS

Versión	Descripción	Autor	Fecha	Comentarios
V0.1	Versión inicial	Iker Larizgoitia	02/10/2007	

TABLA DE CONTENIDOS

1 TESTING EXAMPLE USING JUNIT INSIDE OSGI

This example is based on a simple HelloWorldService which can be downloaded at

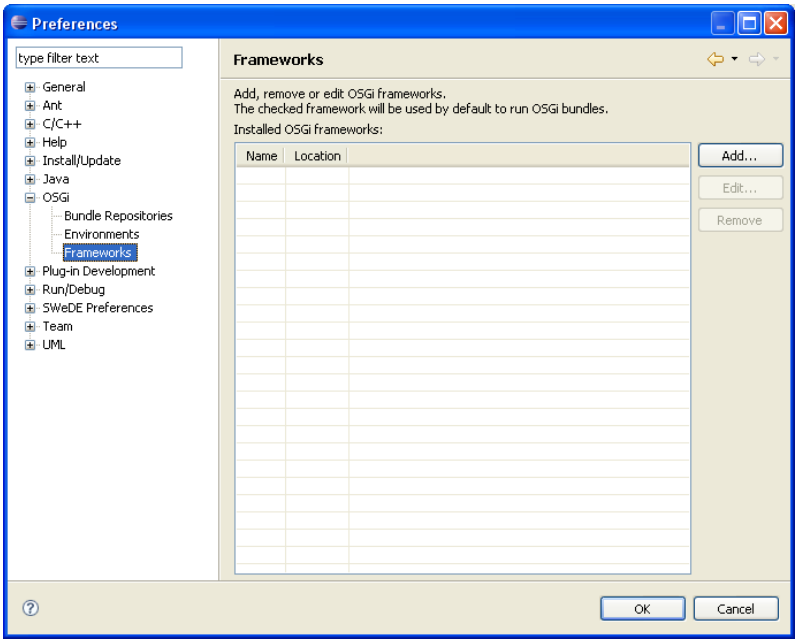
https://130.206.139.226/svn/Smartlab/trunk/examples/TestingJUnit/helloworldtest

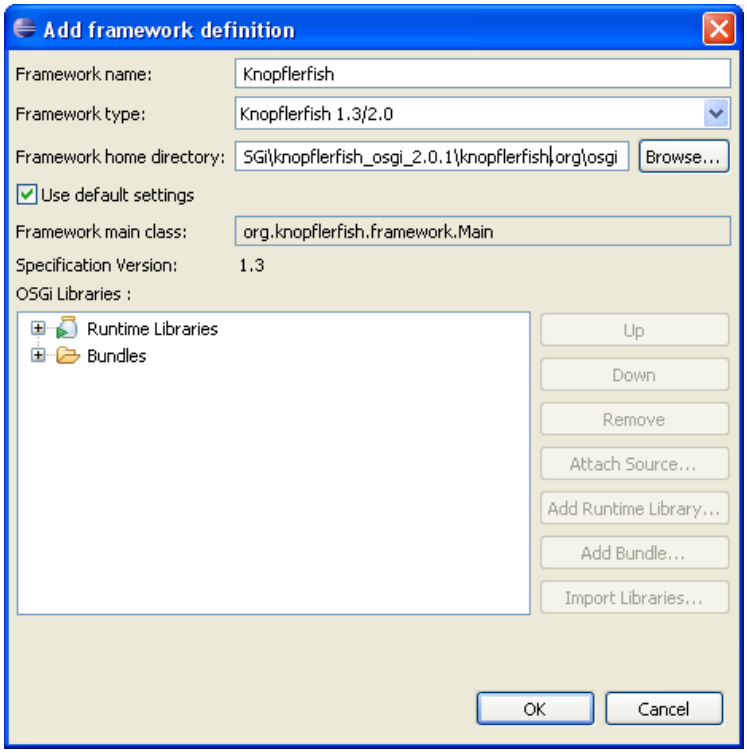
1.1 Configuring Knopflerfish in Eclipse

After installing the Knopflerfish plug-in for Eclipse, you have to configure it properly to be able to launch it from inside Eclipse.

1. Go to the menu *Window* → *Preferences*

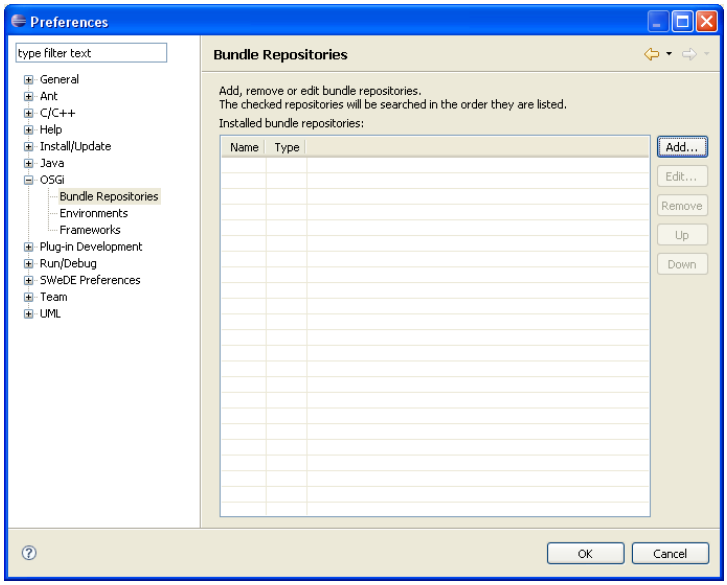
2. Add a new framework configuration

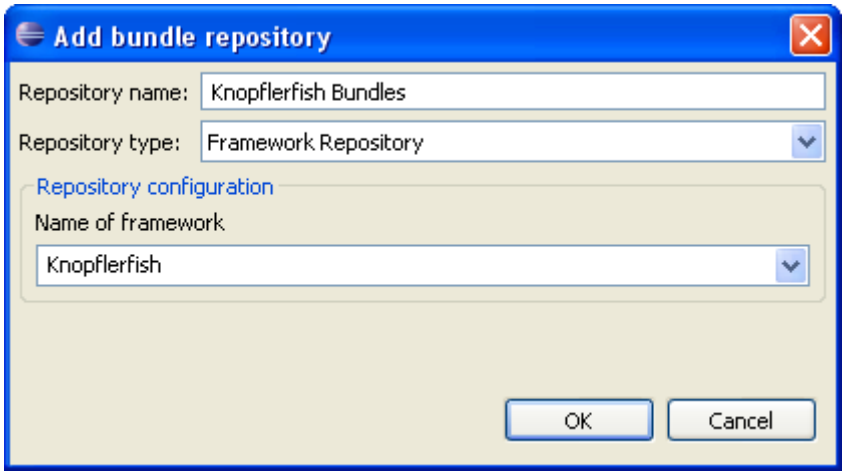




Framework home directory is the *osgi* directory of the Knopflerfish installation.

3. Add a *Bundle Repository*: adding the bundle repository will let you select its bundles later when configuring the framework environment.





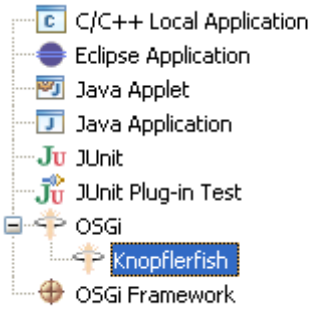
If the *Name of framework* combo does not provide a Knopflerfish entry, close the Window and try again (after configuring the framework).

1.2 Configuring Eclipse to run Knopflerfish

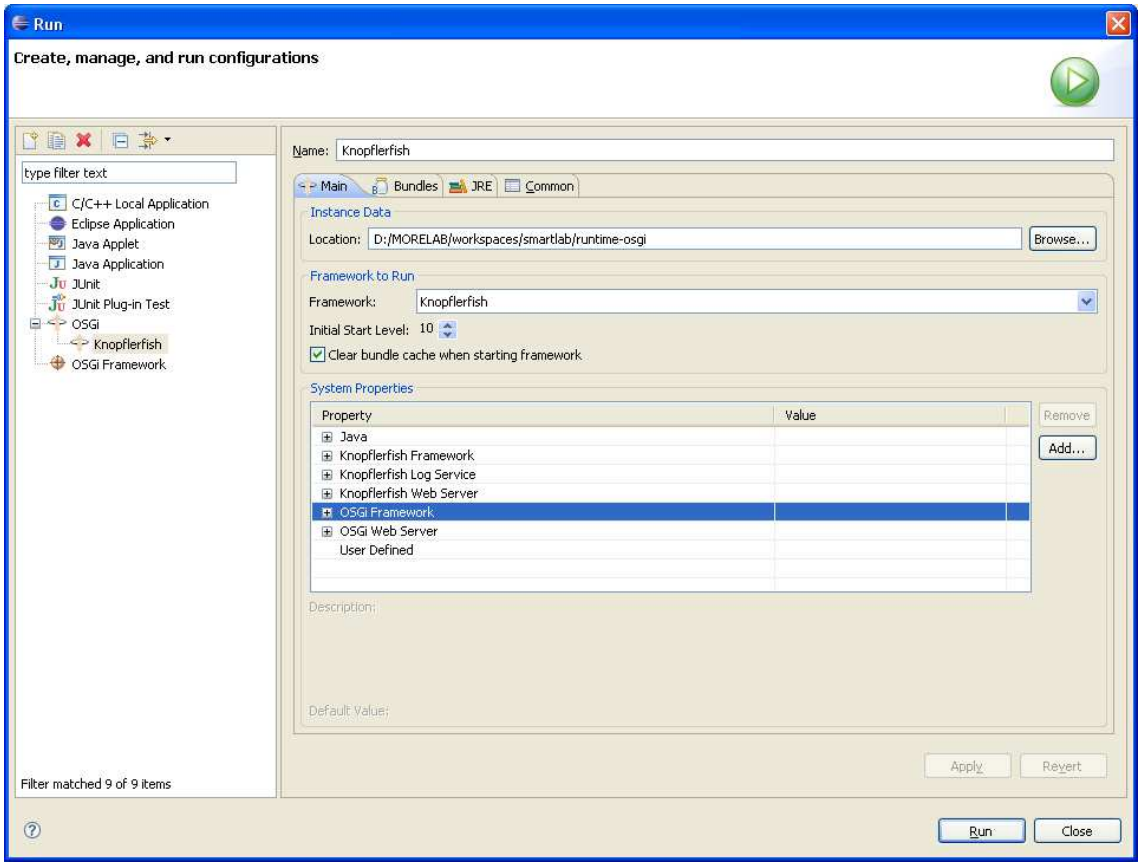
Once the bundles are ready (not covered here, but you can use the build.xml file to compile and generate the bundles):

1. Right-click on the project and *Run→Open Run Dialog…*

2. Select the OSGi option and create a new launch options (call it Knopflerfish).



3. Now in the Main tab configure your framework properties:



Remarks:

- Some of the bundles require the system classloader to be responsible for loading all the classes that are used, but not imported as external packages. This is not a desired behaviour, but some of the bundles of the Knopflerfish distribution need this to work (for example the desktop bundle). For a default execution of the Knopflerfish distribution, a new user defined property must be set with the following value

```
org.osgi.framework.bootdelegation = *
```

- Load the desired bundles from the Bundles tab. This tab contains the bundles included in the Knopflerfish installation and all the bundles available from the workspace. You must add all the bundles you want to be in the OSGi instance. The list of the Knopflerfish default bundles and its init level is (add them in the Bundles Tab):

  initlevel 1

  log_all-2.0.0.jar

  cm_all-2.0.0.jar

  console_all-2.0.0.jar

  component_all-2.0.0.jar

  event_all-2.0.0.jar

initlevel 2

util-2.0.0.jar

crimson-2.0.0.jar

jsdk-2.2.jar

bundlerepository_all-2.0.0.jar

initlevel 3

device_all-2.0.0.jar

useradmin_all-2.0.0.jar

initlevel 4

http_all-2.0.0.jar

initlevel 5

frameworkcommands-2.0.0.jar

logcommands-2.0.0.jar

cm_cmd-2.0.0.jar

consoletty-2.0.0.jar

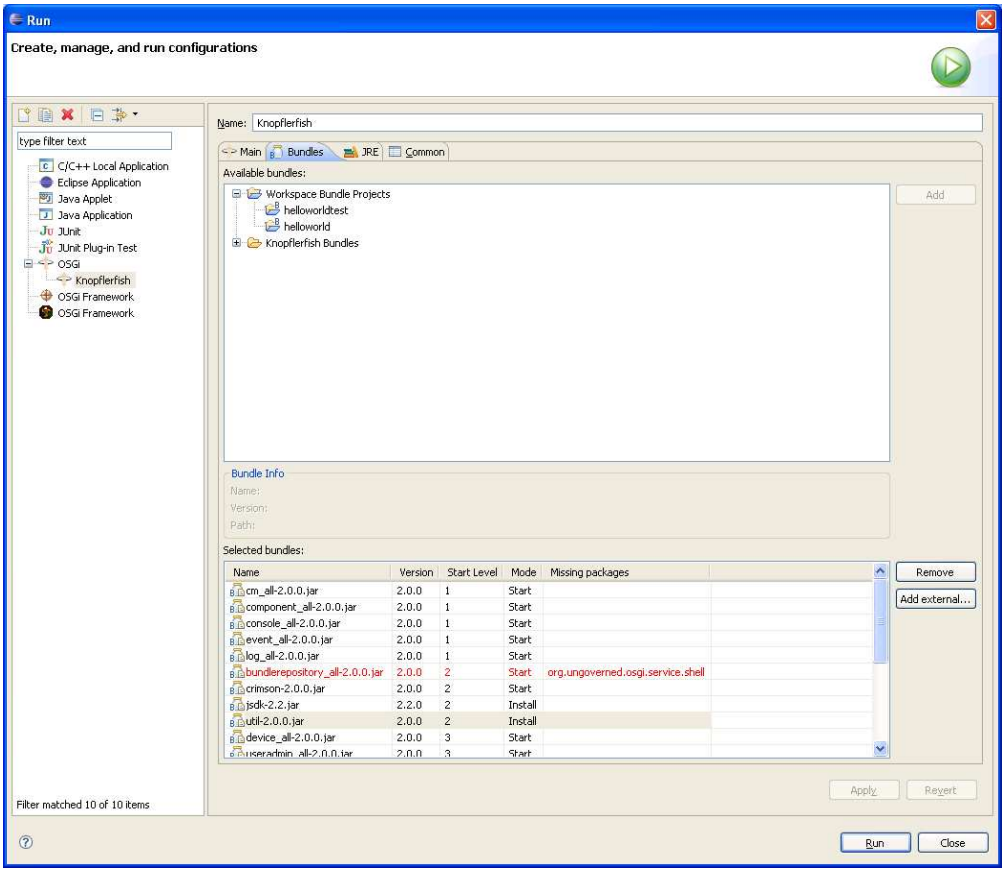
consoletelnet-2.0.0.jar

initlevel 6

desktop_all-2.0.0.jar

initlevel 7

httproot-2.0.0.jar



After adding all the bundles and checked that the dependencies are met, we can Run the framework.

1.3 Testing in OSGi overview

Testing in OSGi (a.k.a. testing in container based systems) is not trivial. The dependency of the framework services and correspondent classes sometimes can make it difficult to test.

There are three possible strategies you can follow to develop your testing code:

1. Have it compiled into the main bundle, but have a build-time option to strip out the test code for deployment purposes.

2. Have it in a fragment, which attaches itself to the main bundle

3. Have it in a bundle that depends on the code being tested (using internal friends etc. if wanting to check non-public API code)

The best and cleanest way is the second one, using fragments, but it's the most complicated to maintain. For some situations, one and three are easier and perfectly valid.

1.4 Using JUnit inside OSGi

Knopflerfish distribution comes with two bundles that help develop JUnit tests and execute them inside the OSGi environment. These bundles are:

junit_all-2.0.0.jar → Contains the JUnit 3 framework classes.
junit_runner_all-2.0.0.jar → Contains a Runner that is able to execute TestCases automatically and dump the results to an XML file.

1.5 Testing Tutorial

The following sections describe how to setup and run the example service and test it inside OSGi. The testing environment in which the example is based is:

- Eclipse Europa
- Knopflerfish OSGi framework
- Knopflerfish plug-in for Eclipse

We are using JUnit 3 conventions to develop our test example. We assume you have a basic knowledge of the JUnit testing framework.

1.5.1 HelloWorld bundle

The bundle we are going to test is a simple HelloWorld Bundle developed with Declarative Services. It exposes a simple method sayHello under the interface HelloWorldService.

You can download the bundle from:

https://130.206.139.226/svn/Smartlab/trunk/examples/TestingJUnit/helloworld

Remarks

The compilation and creation of the bundle is based on a build.xml ant file.

external → contains external libraries used in the bundle (the libraries included here will not

be added to the jar file)

lib → contains internal libraries used in the bundle (the libraries included here will be added to the jar file).

OSGI-INF → it contains the Xml definition of the component used by the Declarative Service to create and configure the services.

OSGI-OPT → it contains any optional resource (usually the documentation and the source code).

1.5.2 HelloWorld test bundle

To enable testing with the JUnit runner provided in the Knopflerfish distribution you have to create the corresponding TestSuites and register them as a Test service in the OSGi ServiceRegistry.

The bundle we are using to test the HelloWorldService has a simple activator which registers the TestSuite in the OSGi environment so that the junit_runner can execute it and generate the results.

```
public void start(BundleContext bc)
  {
      Activator.bc = bc;

    // Sets the name of the TestSuite, needed in the property
    // org.knopflerfish.junit_runner.tests
    TestSuite suiteExists = new TestSuite(HelloWorldExistsTest.class,
"helloworldExists");

    Hashtable props = new Hashtable();
    props.put("service.pid", suiteExists.getName());
    bc.registerService(Test.class.getName(), suiteExists, props);
  }
```

Once the test class is developed, you can configure whether the *junit_runner* should test it or

not in the framework properties. You must set the property

org.knopflerfish.junit_runner.tests

with the name(s) of the test suite(s) to be executed (blank separated names). If you want to automatically exit the framework when the tests are executed, you can set a property called

```
org.knopflerfish.junit_runner.quit = true
```

1.5.3 Executing the Test

From the Run configuration dialog we must add the bundles to the framework configuration and configure the junit runner to start after the bundles to test are ***started***.

JUnit bundles (provided with the Knopflerfish)

junit_all-2.0.0.jar
junit_runner_all-2.0.0.jar

Once the tests have been executed, the results are dumped to a xml file under the framework_dir/junit_grunt