Hello Function – First steps for implementation

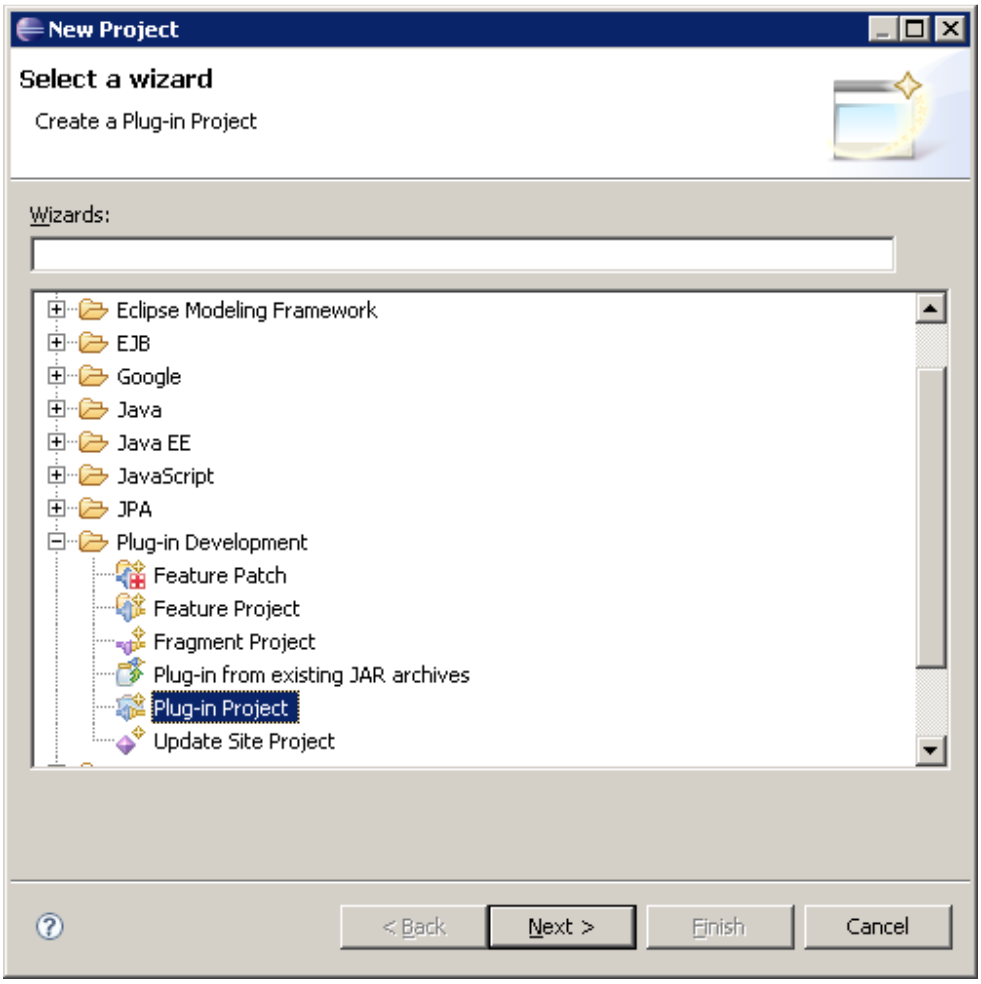
This document describes how to create a bundle with a function running in the secRT. Basically, 4 steps have to be executed:

- Setting up the project
- Implement the function
- Register and activate the function
- Export and test the function

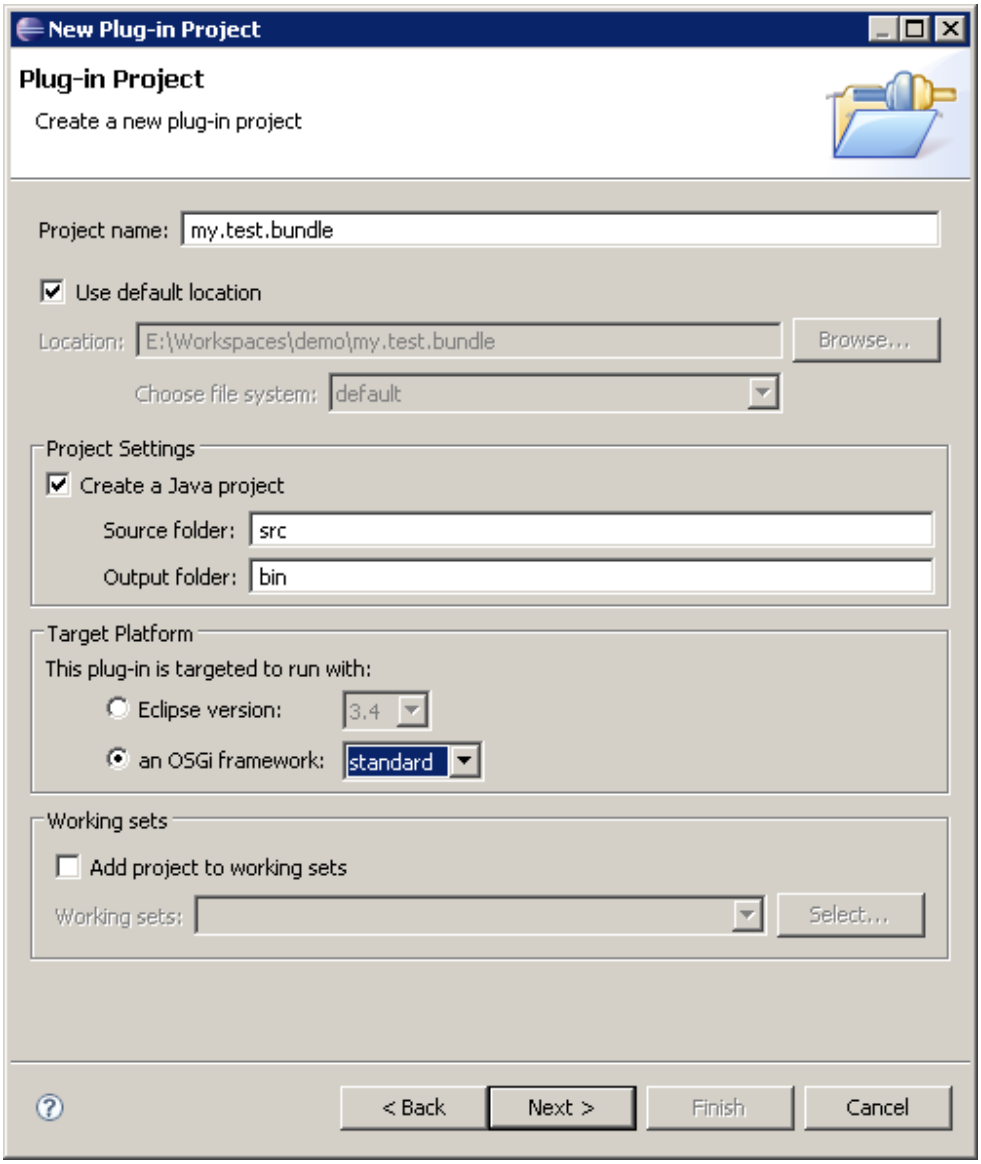
This document guides you through the steps.

Setting up the project

In Eclipse: use the menu item File > New > Project. Select Plug-in-Project.

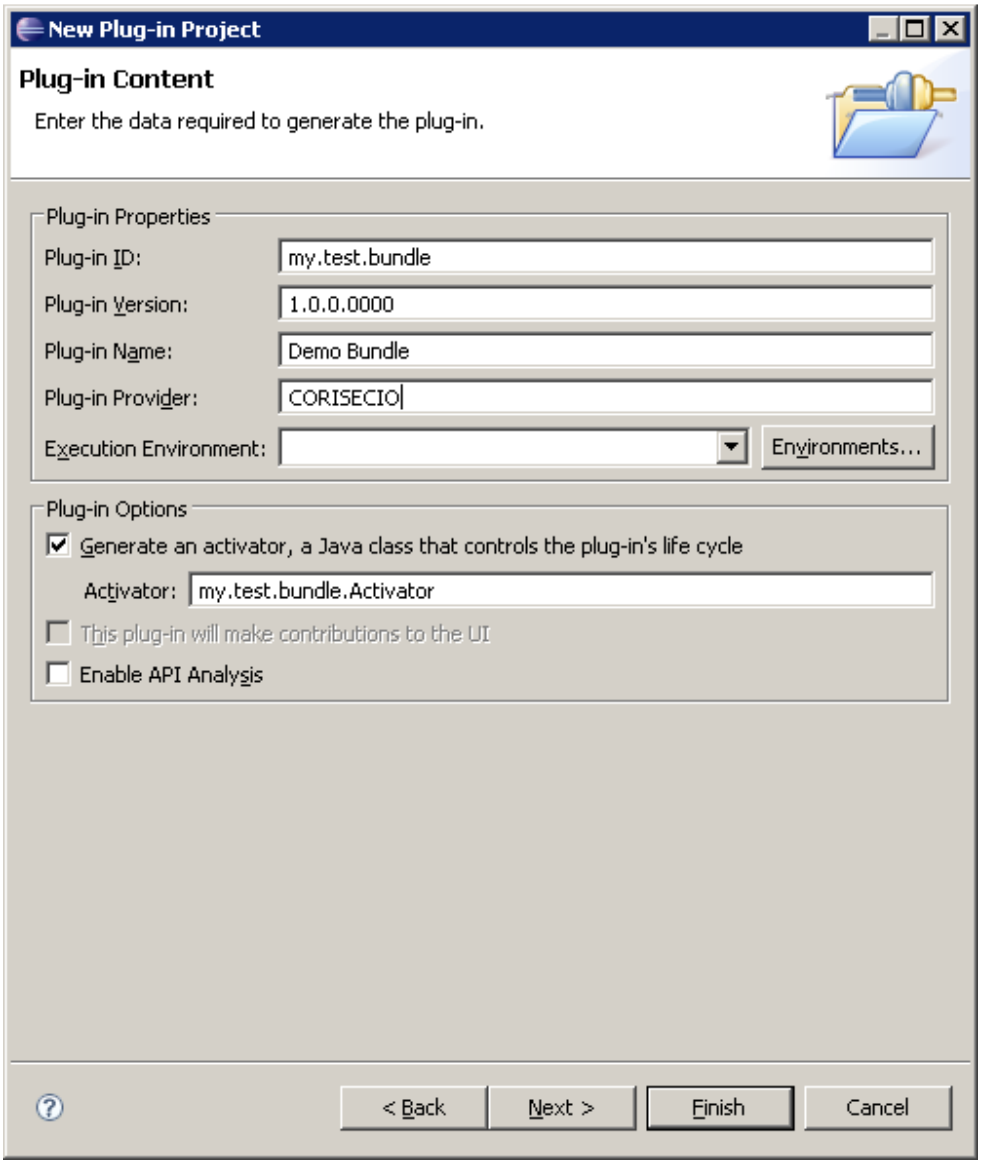


Name the bundle without whitespaces like e.g. my.test.bundle. Choose “standard” as OSGi framework.



Click on Next.

Add four digits to the version number. Define an appropriate Plugin-Name and Plugin-Provider.



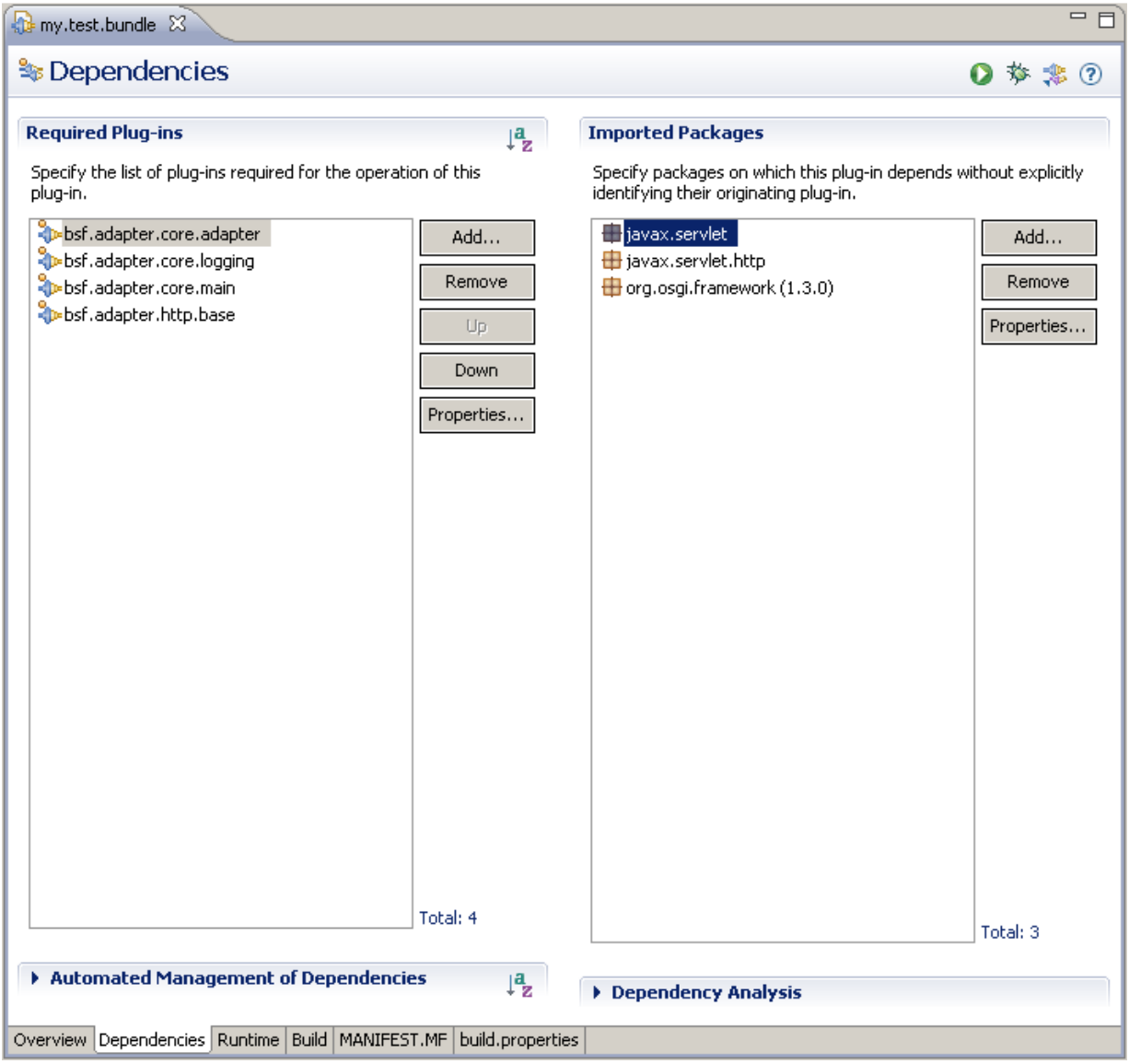
Click on Finish.

The manifest file editor opens. Switch to the tab Dependencies. Click on Add… within Required Plug-ins and select

- Bsf.adapter.core.adapter
- Bsf.adapter.core.logging
- Bsf.adapter.core.main
- Bsf.adapter.http.base

Add the following packages to import by clicking on Add... within Imported Packages and add

- javax.servlet
- javax.servlet.http

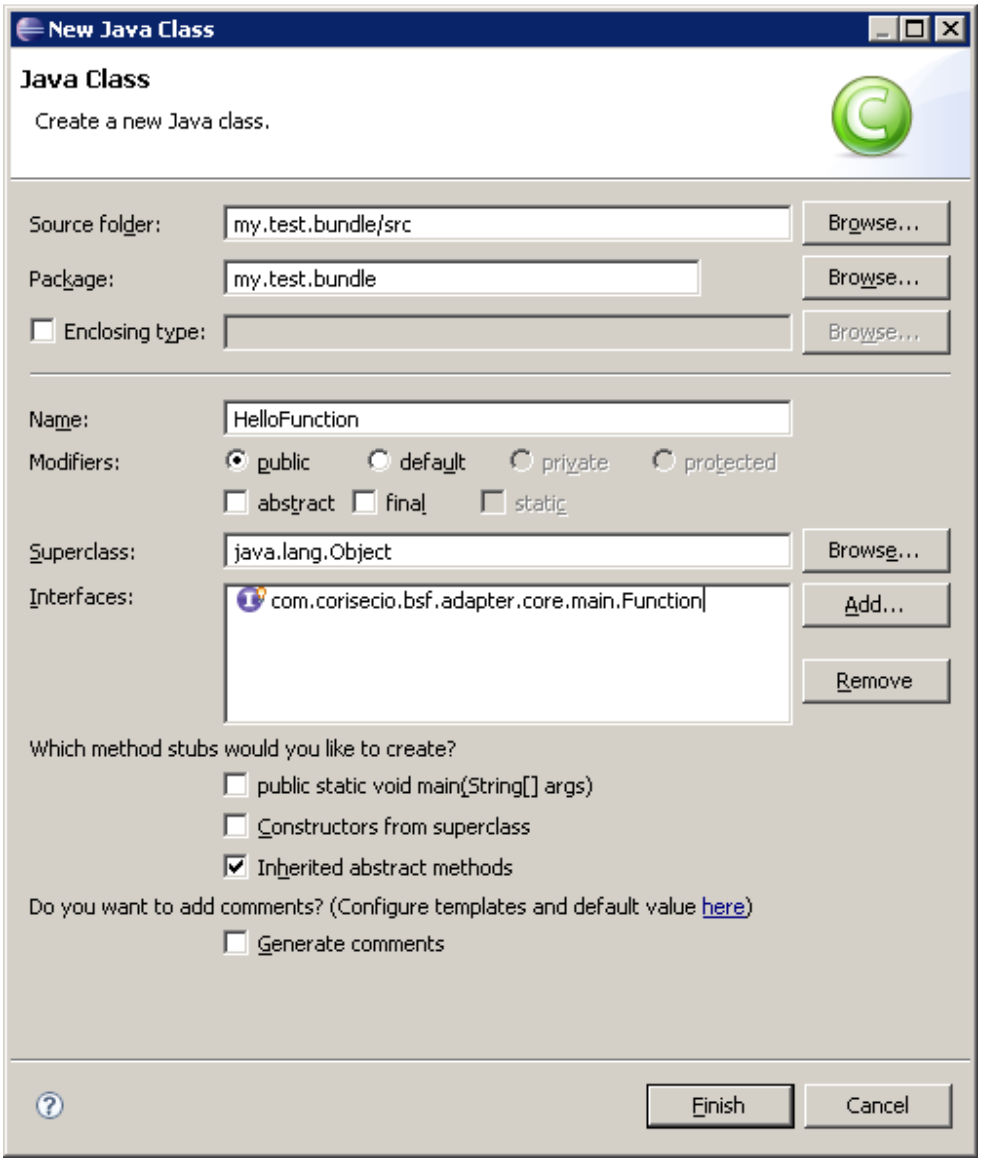


Save the manifest file.

Create a Function

The next step is to implement a Function. We want to return an HTTPResponse with the text “Hello Function”.

To achieve this, create a class called HelloFunction implementing the interface com.corisecio.bsf.adapter.core.main.Function.



Implement the source code as displayed on the screen shot.

HelloFunction.java

```
package my.test.bundle;

import java.util.LinkedList;

import javax.servlet.http.Cookie;

import com.corisecio.bsf.adapter.core.main.BSFObject;
import com.corisecio.bsf.adapter.core.main.Function;
import com.corisecio.bsf.adapter.core.main.FunctionResult;
import com.corisecio.bsf.adapter.core.main.Variables;
import com.corisecio.bsf.adapter.http.base.contents.ContentType;
import com.corisecio.bsf.adapter.http.base.contents.TextPart;
import com.corisecio.bsf.adapter.http.base.data.HTTPResponse;
import com.corisecio.bsf.adapter.http.base.data.HeaderMap;


public class HelloFunction implements Function
{
    public FunctionResult process(BSFObject obj, Variables var) throws Exception
    {
        HeaderMap headers = new HeaderMap();

        return new FunctionResult("ok",
                                  new HTTPResponse("HTTP/1.1",
                                                   headers,
                                                   new TextPart(ContentType.parse("text/html"),
                                                                "<html><body>" +
                                                                "<h1>Hello Function</h1>" +
                                                                "</body></html>"),
                                                   200,
                                                   null,
                                                   new LinkedList<Cookie>()));
    }
}
```

The HeaderMap instance contains the http response headers to be set. Please note that the content type is set by the TextPart to text/html. When 200 is the return code, everything was processed successfully.

Register and activate the function

After successful implementation the function has to be defined.

Configuration data

Although we do not want to have configuration parameters, create a file called my.test.bundle.xsd in the project's META-INF folder. Insert the following lines to the file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```

Here you can define data types for the configuration of your function. For samples, please also see other implementations.

Function definition

As next step we have to define the function. Create a file called adapter.xml in the project's META-INF folder. Insert the following lines to the file:

```
<?xml version="1.0"?>
```

```
<adapter>
  <schema file="META-INF/my.test.bundle.xsd" />

  <functions>
    <function name="Hello Function" class="my.test.bundle.HelloFunction">
      <configuration name="HelloFunctionInstance" type="" />
      <signature inType="BSFObject">
        <out name="ok" outType="HTTPResponse" class="ok" />
      </signature>
    </function>
  </functions>
</adapter>
```

With `<schema file="META-INF/my.test.bundle.xsd" />` we reference the schema file with the data configuration.

In the functions-element all functions of the bundle are defined. Define the function's name in the name-attribute and an id in the class-attribute. The configuration-element defines the name of the created configuration-element (HelloFunctionInstance) and the data type (if empty there is no configuration; otherwise it references to the xsd-file). The signature defines the input type (inType) and a list of possible results with a name and their output types.

Activation and registration of the function

The function is configured by the Workflow editor and nearly the entire configuration is handled by the framework. The bundle activator is responsible for initializing and activating the function correctly. Implement the Activator with the following code:

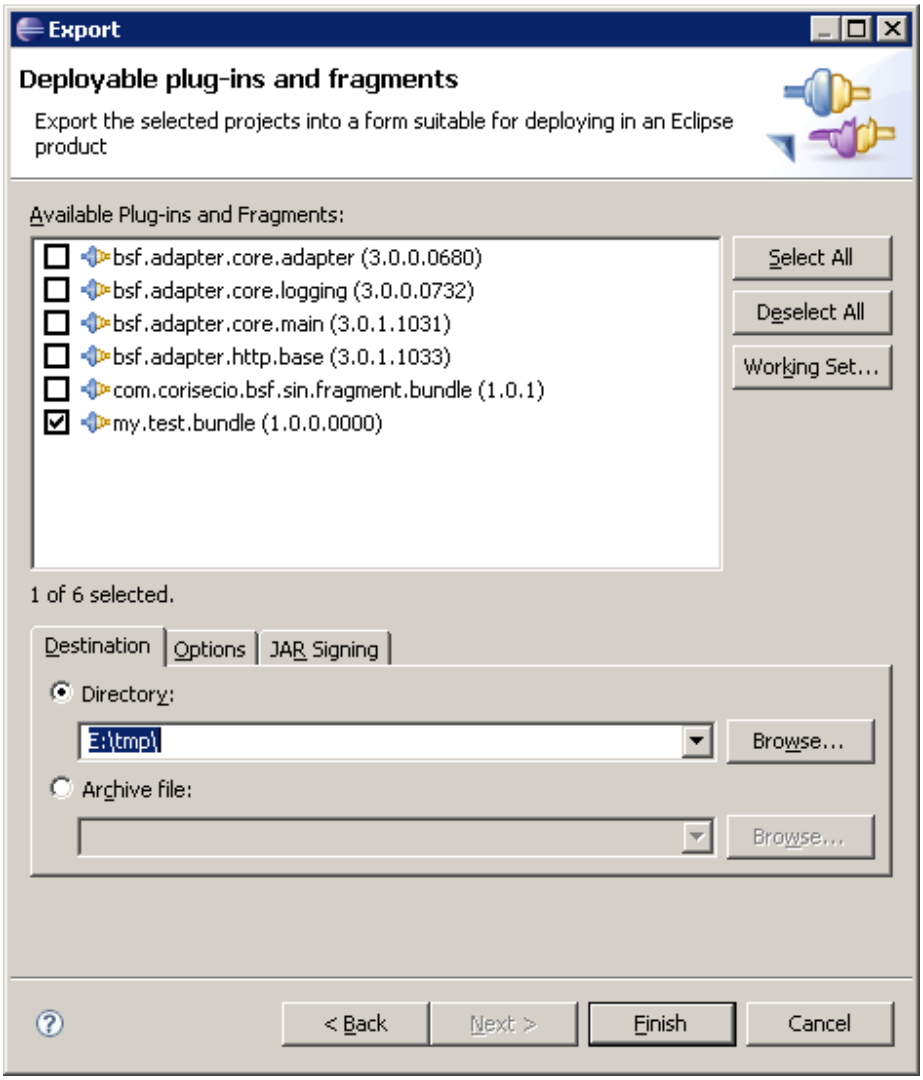
```
package my.test.bundle;

import java.util.Hashtable;

import org.jdom.Element;
import org.osgi.framework.ServiceRegistration;

import com.corisecio.bsf.adapter.core.logging.Logger;
import com.corisecio.bsf.adapter.core.logging.LoggingSourceDescriptor;
import com.corisecio.bsf.adapter.core.main.AbstractBundleActivator;
import com.corisecio.bsf.adapter.core.main.Function;


public class Activator extends AbstractBundleActivator
{
	private final static Logger LOG = new Logger(Activator.class);


	public Activator()
	{
		super("my.test.bundle",
getLoggingSourceDescriptor(Activator.class));
	}

	public void elementAdded(String symbolicName, Element element) throws
Exception
```

```
      {
          if ("HelloFunctionInstance".equals(element.getName()))
          {
            try
            {
              String processorID = element.getAttributeValue("processorID");

              HelloFunction function = new HelloFunction();

              Hashtable<String, String> properties = new Hashtable<String,
String>();
              properties.put("processorID", processorID);

              ServiceRegistration reg =
context.registerService(Function.SERVICE_KEY, function, properties);

              LOG.debug("Registered HelloFunction with id " + processorID);
              myServices.put(processorID, reg);
            }
            catch (Exception ex)
            {
              LOG.error("HTTP proxy not started", ex);
            }
          }
      }

      public static LoggingSourceDescriptor getLoggingSourceDescriptor(final
Class clazz)
      {
            String className = "00";

          if (Activator.class.equals(clazz))
            className = "01";

          return new LoggingSourceDescriptor("01", "test", className);
      }
}
```

The function elementAdded gets the bundles symbolic name and an instance of the configuration element. First the intended function is determined, next the processorID (an internal id needed for the registration in the OSGi framework) is read from the configuration. The instance of the function is created and registered.

Export and test

To export the bundle, open the project context menu. Choose the menu item Export. Select Deployable plug-ins and fragments and check my.test.bundle. Define a destination directory.



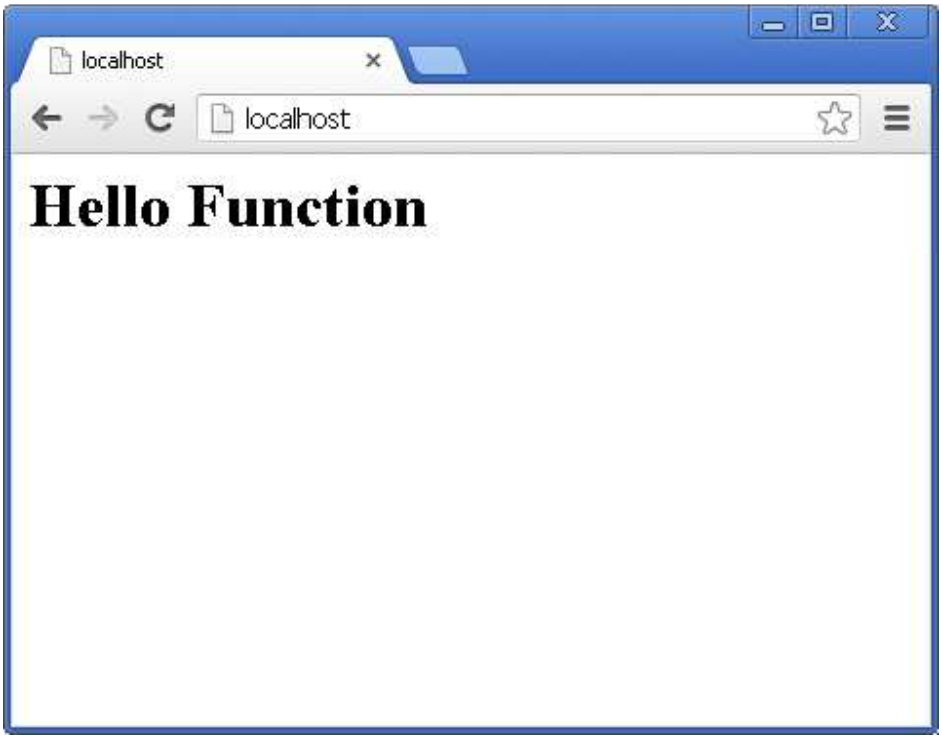
Click on Finish.

The bundle is exported to the plugins-directory within the destination directory. Copy your bundle to the adapter-directory of a deployed gateway and restart the instance.

Log-in to the Admin Console and click on Workflow. Click on New to create a new workflow. Define a name like “HelloFunction workflow”. Click on Ok, check “HelloFunction workflow” and click on Edit. Click on Configure right next to the drop-down-box with App listener #1. Enter a port number like e.g. 80 and click on Ok. Choose the entry “Hello Function” in the drop-down-box below Function and click on Add. Click on Ok. Check “HelloFunction workflow” and click on “Activate/Deactivate”.

Open a browser and enter the url [http://[host-of-the-gateway]/](http://[host-of-the-gateway]/)

Now, the implementation is completed.


localhost
localhost
Hello Function