OSGi with Eclipse Equinox - Tutorial

Lars Vogel

Version 2.2

06.02.2011

Revision History		
Revision 0.1 Created	03.09.2007	Lars Vogel
Revision 0.2 - 2.3 bugfixes and enhancements	25.10.2008 -06.02.2011	Lars Vogel

OSGi with Eclipse Equinox

This tutorial gives an overview of OSGi. It explains the creation and consumption of OSGi services via ServiceTrackers and declarative services. Eclipse Equinox is used as an standalone OSGi server. For this tutorial Eclipse 3.6 (Helios) is used.

Table of Contents

1. OSGi

1.1. Overview

OSGi is a specification of a service and module platform in Java at runtime. The core of the OSGi specification defines a component and service model. This component model allows to defines components and services and provides the means to activate, de-activate, update and de-install them dynamically.

The smallest unit of modularization in OSGi is a bundle. OSGi defines a registry which bundles can use to publish services or register to other services.

1.2. Key features

The key features of OSGi are:

- Modularization
- Runtime Dynamic
- Service Orientation

In the authors personal opinion the strongest feature of OSGi is that you can define which package of your Java Projects should be visible to other Java projects. This way you can effectively control which Java classes in these projects can be used, e.g. define your API.

1.3. Implementations

OSGi has several implementations, for example Knopflerfish OSGi or Apache Felix. Eclipse Equinox is currently the reference implementation of the OSGi specification.

Eclipse Equinox is the runtime environment on which the Eclipse IDE and Eclipse RCP application are based. In Eclipse the smallest unit of modularization is a plugin . The terms plugin and bundle are (almost) interchangable. An Eclipse plugin is also an OSGi bundle and vice versa. Eclipse Equinox extends the concept of bundles with the concept of extension points .

1.4. OSGi bundles

The OSGi specification defines the OSGi bundle as the unit of modularization. A bundle is a cohesive, self-contained unit, which explicitly define its dependencies to other modules / services and explicitly defines its external API. OSGi bundles are .jar files with additional meta information. This meta information is stored in the folder "META-INF" in the file "MANIFEST.MF". MANIFEST.MF is part of a standard jar specification. Any non-OSGI runtime will ignore the OSGI metadata. Therefore OSGi bundles can be used without restriction in non-OSGi Java environments.

Each bundle has a symbolic name which is defined via the property "Bundle-SymbolicName" in the MANIFEST.MF. Convention is that this name starts with the reverse domain name of the author of the bundle, e.g. "de.vogella.myfirstbundle".

Each bundle has also a version number in the property "Bundle-Version". This version number and the symbolic name uniquely identify a bundle in OSGi. The OSGi runtime can load the same bundle with different version numbers.

1.5. Bundle dependencies and public API

Via MANIFEST.MF a bundle can define its dependency to other bundles and services. A bundle can define that it depends on a certain version (or a range) of another bundle, e.g. bundle A can define that it depends on bundle C in version 2.0, while bundle B defines that it depends on version 1.0 of bundle C.

If a class wants to use a class from another bundles this dependency must be defined in the MANIFEST.MF, otherwise you will receive a ClassNotFound Exception. This restriction is enforced via a specific OSGi classloader.

MANIFEST.MF also defines the Java classes which should be available to other bundles as public API. This definition is done based on the packages name. Classes which are not exported via the MANIFEST.MF are not visible to other bundles. This restriction is enforced in OSGi via a special Java class loader. Access to the restricted classes is not possible, also not via reflection.

1.6. OSGi services

A bundles can register and use services in OSGi. OSGi provides therefore a central registry for this purpose. A service is defined by a Java interface (POJI - plain old Java interface).

Access to the service registry is performed via the class BundleContext. OSGi injects the BundleContext into each bundle during the startup of the bundle. A bundle can also register itself to the BundleContext ServiceEvents which are for example triggered if a new service is installed or de-installed.

1.7. OSGi dependency management

OSGi is responsible for the dependency management between the bundles. These dependencies can be divided into:

- Bundle (Package) Dependencies
- Service Dependencies

OSGi reads the manifest.mf of a bundle during the installation of the plugin and ensures that all dependent bundles are also loaded if the bundle is activated. If the dependencies are not meet then the bundle is not loaded. Bundle / package dependencies are based on dependencies between standard Java objects and in case OSGi can not resolve all dependencies this results in a ClassNotFoundException.

As service in OSGi can be dynamically started and stopped, therefore the bundles must manage these dependencies themselves. The bundles can use the service listeners to get informed if a services is stared or stopped.

1.8. Bundle Life cycle

With the installation of a bundle in the OSGi runtime this bundle is persisted in a local bundle cache. The OSGi runtime is then trying to resolve all dependencies of the bundle. If all required dependencies are resolved the bundle is in the status "RESOLVED" otherwise it is in the status "INSTALLED". If several bundles exists which would satisfy the dependency then the bundle with the highest version is taking. If the version are the same then the bundle with the lowest ID is taken. If the bundle is started its status is "STARTING". Afterwards it is "ACTIVE".

1.9. OSGi MANIFEST.MF example

The following is an example of a MANIFEST.MF.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Popup Plug-in
Bundle-SymbolicName: de.vogella.rcp.intro.commands.popup; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: de.vogella.rcp.intro.commands.popup.Activator
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

Via the "Bundle-RequiredExecutionEnvironment" statement a bundle can specify which Java version is required to run the bundle. If this requirement is not fulfilled then the OSGi runtime does not load the bundle.

2. Installation

You need to have Java installed. I recommended to use Java 1.6.

Download Eclipse from the http://www.eclipse.org/ and unpack it to any directory. No installation procedure is required.

3. OSGI console

We will later use the OSGI console. This console is like a MS-DOS prompt. In this console you can type command to perform certain OSGI actions. For following is a reference of OSGi commands.

Use for example the command ss to get an overview of all bundles and their status.

Table 1. OSGi commands

Command	Desription

help	Lists the available commmands.
ss	Gives you an overview of the installed bundles and their status.
ss vogella	Gives you an overview of the bundles and their status if they have vogella within their name.
start id	Starts the bundle with id
stop id	Stops the bundle with id
install URL	Installs a bundles from an URL
uninstall id	Uninstalls the bundle with id
bundle "bundleid""	Show information about the bundle, including the registered and used services.
services filter	Show all available services and their consumer. Filter is an optional LDAP filter, e.g. to see all services use "services (objectclass="ManagedService)".

Bundles can be identified via their id which is displayed by the command ss.

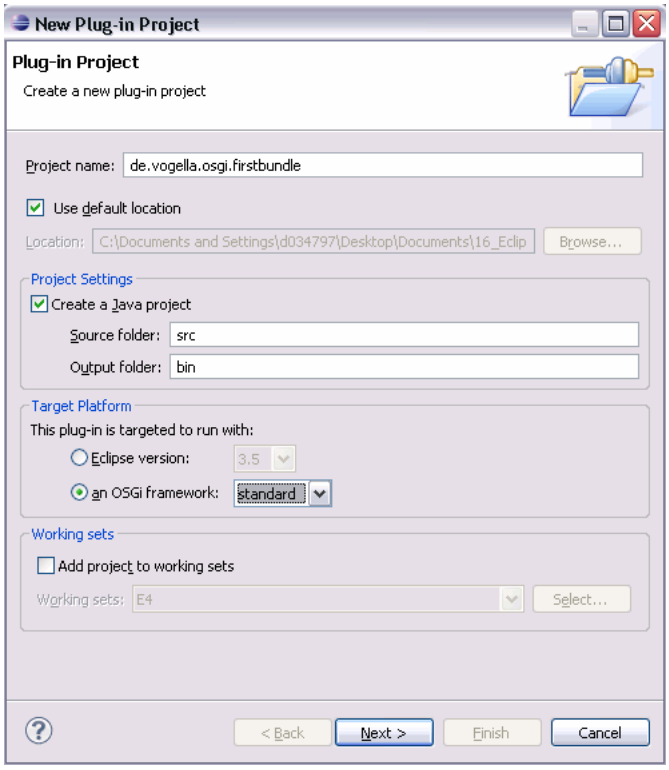
You can also access the OSGi console of your running Eclipse IDE . In the "Console" of your running Eclipse you find a menu entry with the tooltip "Open Console". If you select here "Host OSGi Console" you have access to your running OSGi instance. To see all available commands in the OSGi console type "help".

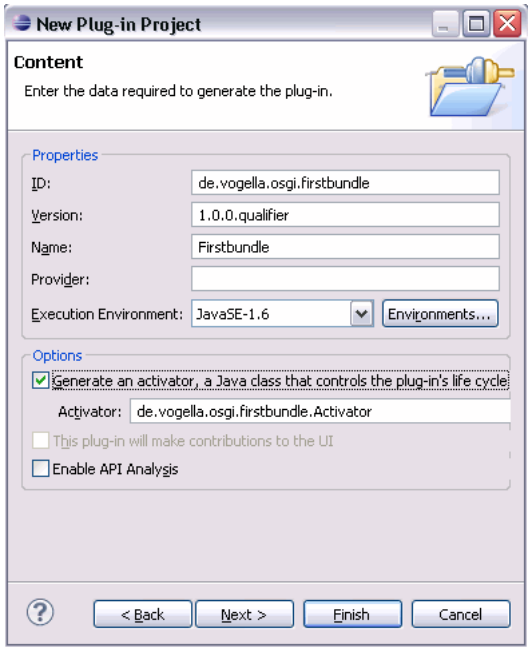
4. Your first OSGi bundle

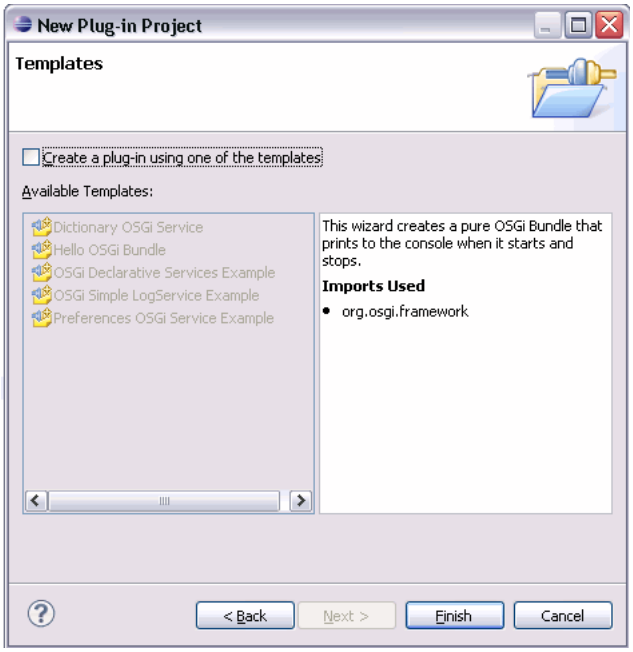
The following will create a simple OSGi bundle and run it within Eclipse. At the end of this chapter you will also export your bundle to use it later in a standalone OSGi server.

4.1. Create

Create a new plugin project "de.vogella.osgi.firstbundle".







4.2. Coding

Create the following thread class.

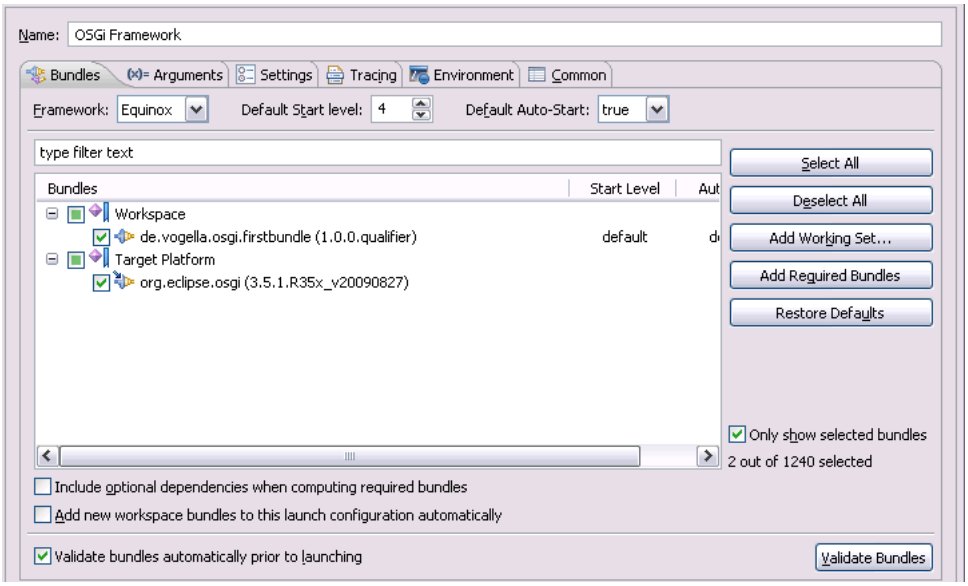
```
package de.vogella.osgi.firstbundle.internal;

public class MyThread extends Thread {
	private volatile boolean active = true;

	public void run() {
		while (active) {
			System.out.println("Hello OSGI console");
			try {
				Thread.sleep(5000);
			} catch (Exception e) {
				System.out.println("Thread interrupted " + e.getMessage());
			}
		}
	}

	public void stopThread() {
		active = false;
	}
}
```

Change the class Activator.java to the following.

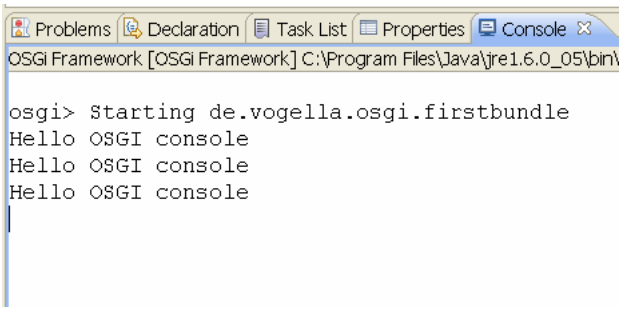
```
package de.vogella.osgi.firstbundle;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

import de.vogella.osgi.firstbundle.internal.MyThread;

public class Activator implements BundleActivator {
	private MyThread myThread;

	public void start(BundleContext context) throws Exception {
		System.out.println("Starting de.vogella.osgi.firstbundle");
		myThread = new MyThread();
		myThread.start();
	}


	public void stop(BundleContext context) throws Exception {
		System.out.println("Stopping de.vogella.osgi.firstbundle");
		myThread.stopThread();
		myThread.join();
	}

}
```

4.3. Run

Select your manifest.mf, right-click, select Run As-> Run Configuration. Create a OSGi Framework launch configuration. Deselect all bundles except your de.vogella.osgi.firstbundle. Press then "Add Required bundles".

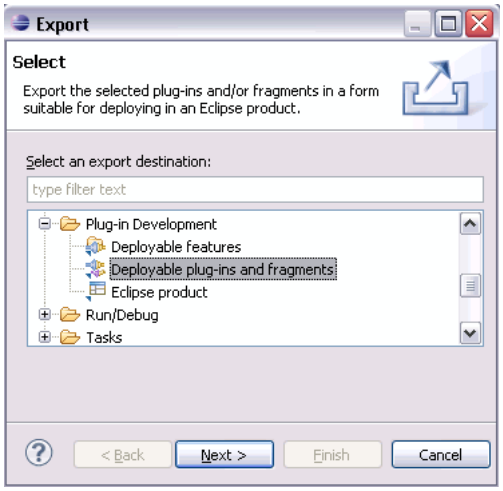


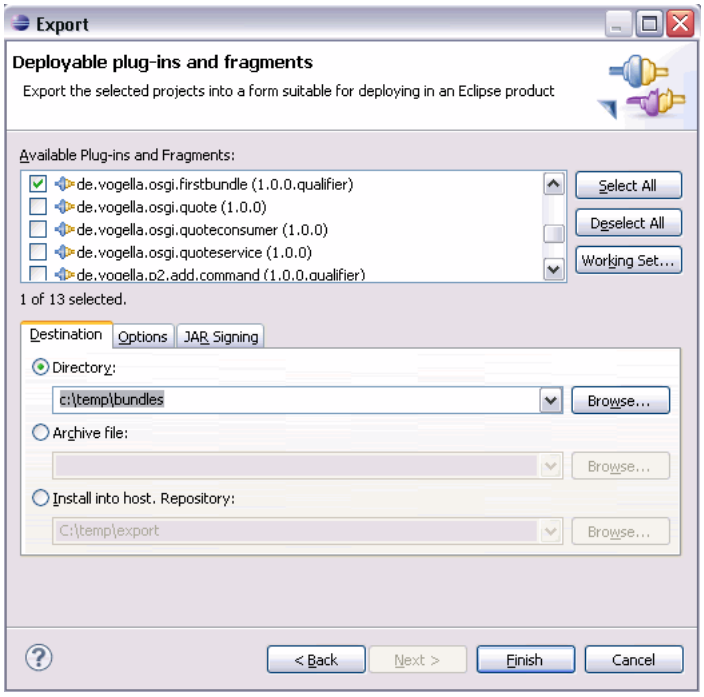
Run this configuration. This should give you the following output. Every 5 second a new message should be written to the console.



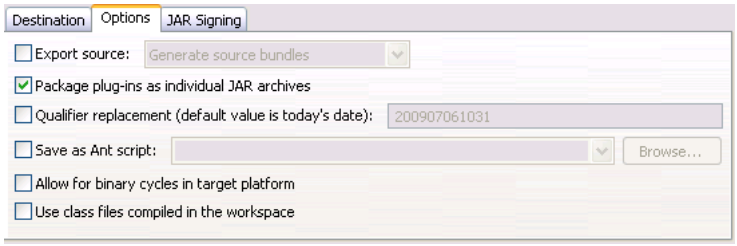
4.4. Export your bundle

Export your bundle. This will allow you to install it into a OSGi runtime. Select your bundle and choose File -> Export -> Plug-in Development -> "Deployable plug-ins and fragment".





Unflag the option to export the source.



5. Running a standalone OSGI server

After running and creating bundles in Eclipse this chapter will show how to run Equinox as a OSGI standalone runtime. .

In your Eclipse installation directory identify the file org.eclipse.osgi*.jar. This file should be in the "plugin" folder. Copy this jar file to a new place, e.g. c:\temp\osgi-server. Rename the file to "org.eclipse.osgi.jar".

Start your OSGI server via the following command.

```
java -jar org.eclipse.osgi.jar -console
```

You can use "install URL" to install a bundle from a certain URL. For example to install your bundle from "c:\temp\bundles" use:

```
install file:c:\temp\bundles\plugins\de.vogella.osgi.firstbundle_1.0.0.jar
```

> You properly need to correct the path and the bundle name on your system.

You can start then the bundle with start and the id.

```
C:\temp\osgi-server>java -jar org.eclipse.osgi_3.4.2.R34x_v20080826-1230.jar -console

osgi> install file:c:\temp\bundles\plugins\de.vogella.osgi.firstbundle_1.0.0.jar
Bundle id is 1

osgi> start 1
Starting de.vogella.osgi.firstbundle

osgi> Hello OSGI console
```

> You can remove all installed bundles with the -clean parameter.

6. Define a Service and service consumption

6.1. Overview

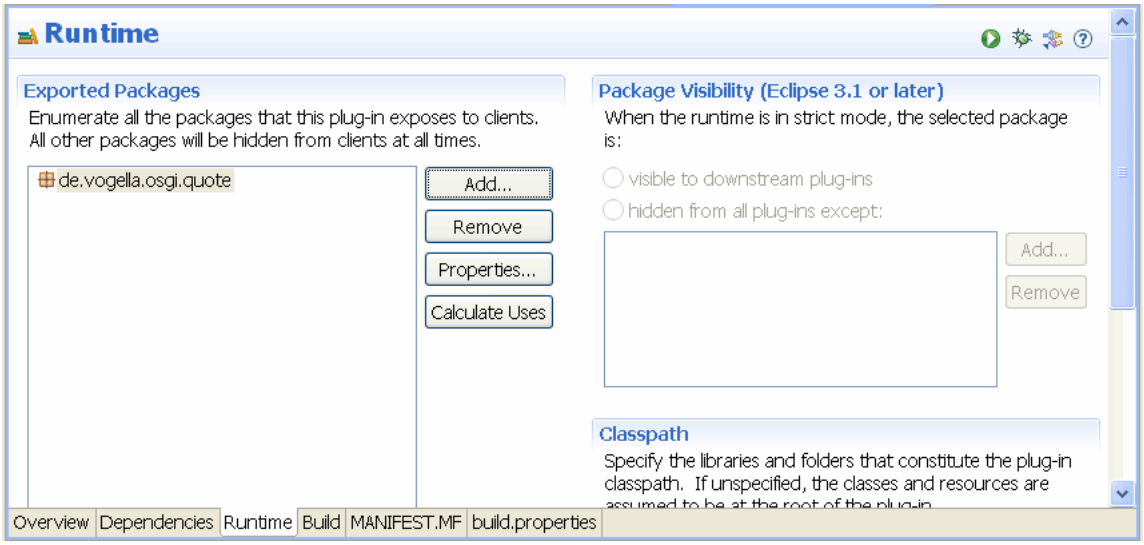
OSGi platform provides a flexible mechanism for provisioning functionality via services. In the following we will define and consume a service. Our service will return "famous quote".

6.2. How to build services

A service in OSGi is defined by a standard Java class or interface. It is common practice to define the service via a bundle which only contains the interface definition. This allows to change the implementation of the service via a different bundle.

6.3. Define the service interface

Create a plugin project "de.vogella.osgi.quote" and the package "de.vogella.osgi.quote". Do not use a template. You do not need an activator. Afterwards select the MANIFEST.MF and the "Runtime" tab. Add "de.vogella.osgi.quote" to the exported packages.



Create the following interface "IQuoteService".

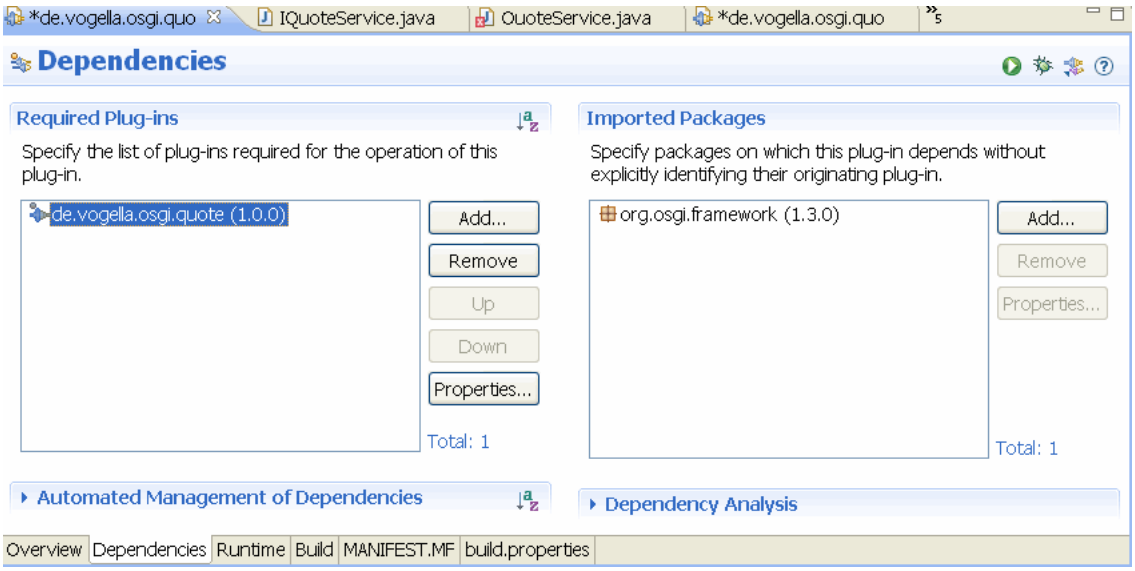
```
package de.vogella.osgi.quote;

public interface IQuoteService {
        String getQuote();
}
```

6.4. Create service

We will now define a bundle which will provide the service.

Create a plugin project "de.vogella.osgi.quoteservice". Do not use a template.

Select the MANIFEST.MF and dependecy tab. Add "de.vogella.osgi.quote" to the required plugins.



Create the following class "QuoteService".

```
package de.vogella.osgi.quoteservice.internal;

import java.util.Random;

import de.vogella.osgi.quote.IQuoteService;

public class QuoteService implements IQuoteService {

        @Override
        public String getQuote() {
                Random random = new Random();
                // Create a number between 0 and 2
                int nextInt = random.nextInt(3);
                switch (nextInt) {
                case 0:
                        return "Tell them I said something";
                case 1:
                        return "I feel better already";
                default:
                        return "Hubba Bubba, Baby!";
                }

        }
}
```

Register the service in the class Activator.

```
package de.vogella.osgi.quoteservice;

import java.util.Hashtable;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

import de.vogella.osgi.quote.IQuoteService;
import de.vogella.osgi.quoteservice.internal.QuoteService;

public class Activator implements BundleActivator {

        public void start(BundleContext context) throws Exception {
                IQuoteService service = new QuoteService();
                // Third parameter is a hashmap which allows to configure the service
                // Not required in this example
                context.registerService(IQuoteService.class.getName(), service,
                                null);
                System.out.println("IQuoteService is registered");
        }

        public void stop(BundleContext context) throws Exception {
        }
}
```

6.5. Install service bundles

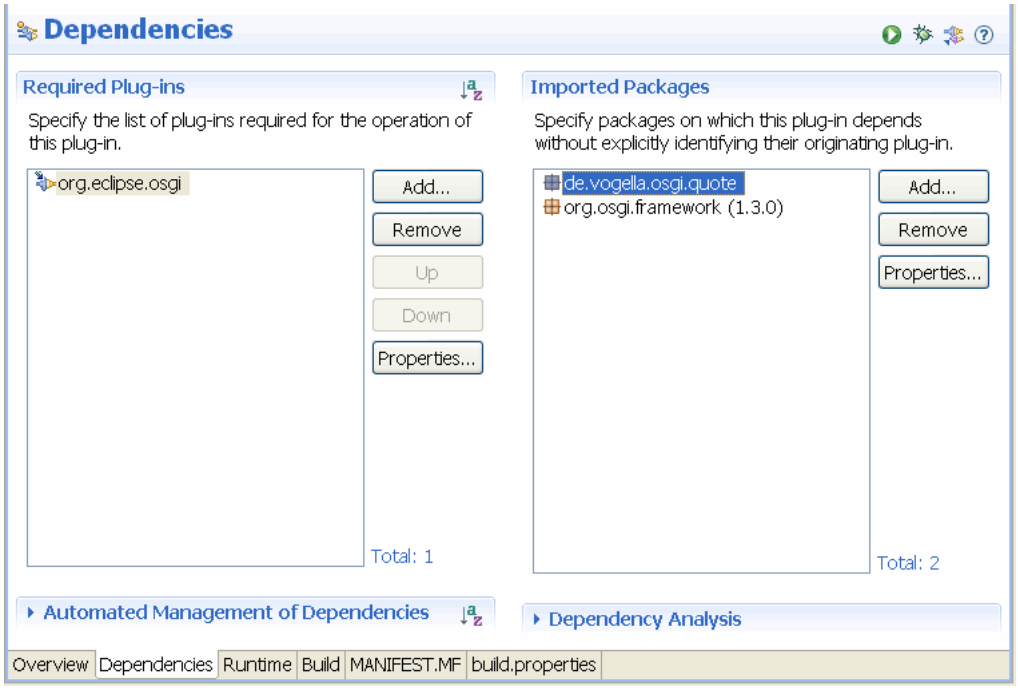
Export your bundles and install them on your server. Start the service bundle.

```
osgi> install file:c:\temp\bundles\plugins\de.vogella.osgi.quote_1.0.0.jar
Bundle id is 3

osgi> install file:c:\temp\bundles\plugins\de.vogella.osgi.quoteservice_1.0.0.jar
Bundle id is 4

osgi> start 4
IQuoteService is registered

osgi>
```

Nothing fancy happens, as we are not yet consuming our service.

6.6. Use your service

Create a new plugin "de.vogella.osgi.quoteconsumer". Add also a dependency to the package "de.vogella.osgi.quote".



Please note that we have added the dependency against the package NOT against the plugin. This way we later replace the service with a different implementation.

Lets register directly to the service and use it.

```
package de.vogella.osgi.quoteconsumer;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;

import de.vogella.osgi.quote.IQuoteService;

public class Activator implements BundleActivator {

	private BundleContext context;
	private IQuoteService service;

	public void start(BundleContext context) throws Exception {
		this.context = context;
		// Register directly with the service
		ServiceReference reference = context
				.getServiceReference(IQuoteService.class.getName());
		service = (IQuoteService) context.getService(reference);
		System.out.println(service.getQuote());
	}

	public void stop(BundleContext context) throws Exception {
		System.out.println(service.getQuote());
	}

}
```

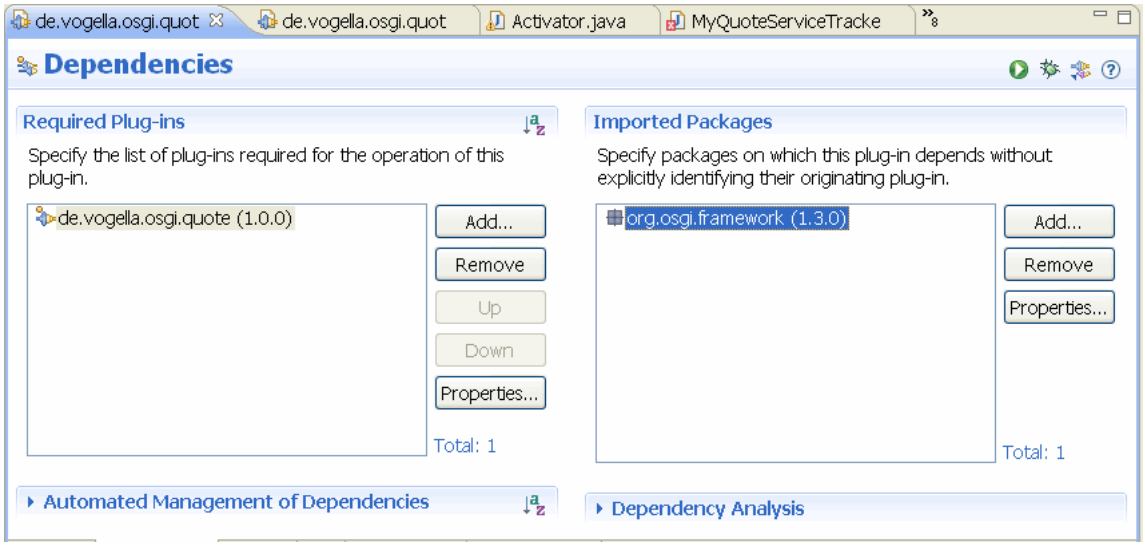
Export this bundle, install it and start and stop it. Everything work. But if you stop the service bundle then your receive an error.

```
osgi> install file:c:\temp\bundles\plugins\de.vogella.osgi.quoteconsumer_1.0.0.jar
Bundle id is 5

osgi> start 5
Tell them I said something

osgi> stop 5
Tell them I said something

osgi> stop 4

osgi> stop 5

osgi> start 5
org.osgi.framework.BundleException: Exception in de.vogella.osgi.quoteconsumer.Activator.start() of bundle
gi.quoteconsumer.
        at org.eclipse.osgi.framework.internal.core.BundleContextImpl.startActivator(BundleContextImpl.java
        at org.eclipse.osgi.framework.internal.core.BundleContextImpl.start(BundleContextImpl.java:984)
        at org.eclipse.osgi.framework.internal.core.BundleHost.startWorker(BundleHost.java:346)
```

The reason for this is that OSGi is a very dynamic environment and service may be registered and de-registered any time. The next chapter will use a service tracker to improve this.

6.7. Use your service with a service tracker

OSGi provides a service to track service events (new service / stop services / started service). For this the package "org.osgi.util.tracker" is used. This is part of the org.eclipse.osgi plugin.



To use this define the following class "MyQuoteServiceTrackerCustomizer"

```
package de.vogella.osgi.quoteconsumer;

import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
import org.osgi.util.tracker.ServiceTrackerCustomizer;

import de.vogella.osgi.quote.IQuoteService;

public class MyQuoteServiceTrackerCustomizer implements
		ServiceTrackerCustomizer {

	private final BundleContext context;

	public MyQuoteServiceTrackerCustomizer(BundleContext context) {
		this.context = context;
	}

	private MyThread thread;

	@Override
	public Object addingService(ServiceReference reference) {
		IQuoteService service = (IQuoteService) context.getService(reference);
		thread = new MyThread(service);
		thread.start();
		return service;
	}

	@Override
	public void modifiedService(ServiceReference reference, Object service) {
		// removedService(reference, service);
		// addingService(reference);
	}

	@Override
	public void removedService(ServiceReference reference, Object service) {
		context.ungetService(reference);
		System.out.println("How sad. Service for quote is gone");
		thread.stopThread();
	}

	public static class MyThread extends Thread {

		private volatile boolean active = true;
		private final IQuoteService service;

		public MyThread(IQuoteService service) {
			this.service = service;
		}

		public void run() {
			while (active) {
				System.out.println(service.getQuote());
				try {
					Thread.sleep(5000);
				} catch (Exception e) {
					System.out.println("Thread interrupted " + e.getMessage());
				}
			}
		}

		public void stopThread() {
			active = false;
		}
	}

}
```

You also need to register a service tracker in your activator of your serviceconsumer.

```
package de.vogella.osgi.quoteconsumer;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.util.tracker.ServiceTracker;

import de.vogella.osgi.quote.IQuoteService;

public class Activator implements BundleActivator {

        private ServiceTracker serviceTracker;

        public void start(BundleContext context) throws Exception {
                System.out.println("Starting quoteconsumer bundles");
                // Register directly with the service
                MyQuoteServiceTrackerCustomizer customer = new MyQuoteServiceTrackerCustomizer(
                                context);
                serviceTracker = new ServiceTracker(context, IQuoteService.class
                                .getName(), customer);
                serviceTracker.open();
        }

        public void stop(BundleContext context) throws Exception {
                System.out.println("Stopping quoteconsumer bundles");
                serviceTracker.close();
        }

}
```

Export your bundle again. Start the OSGI console. Use the update command or the install command to get the new version of your bundle and start it. Once you start your service the tracker will be called and the consumer bundle will start writing messages to the console. Stop the service and verify that the consumer does not use the service anymore.

6.8. Problems with service tracker

The problem with service trackers it they still obey Java rules. In case your service consumer keeps a reference of the service, this service can not get removed via the OSGi framework. The other disadvantage is that the service tracker require a lot of boilerplate code. To solve these issues declarative services were developed.

7. Declarative Services

7.1. Overview

Declarative services (DS) allow to define and consume services via metadata (XML). Via DS you can define OSGi services without having any dependency to the OSGi platform, e.g. services can be defined as POJO's. This allows that these services can be tested independently of the OSGi runtime.

The "OSGi service component" is responsible for starting the service (service component). For the service consumer it is not visible if the service has been created via declarative service or via other means.

Service Components consists out of a XML description (Component Description) and an object (Component Instance). The component description contains all information about the service component, e.g. the class name of the component instance and the service interface.

A reference to the component description is maintained in the "MANIFEST.MF" file. This file is read by the OSGi runtime and if a component description is found the corresponding service is created. The component description in "MANIFEST.MF" looks for example like the following.

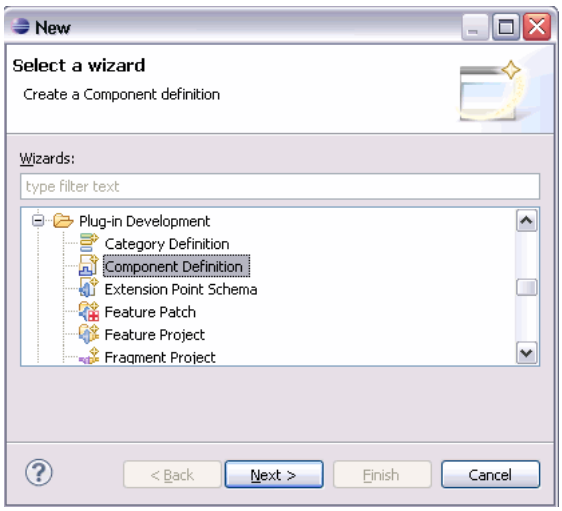
```
Service-Component: component.xml, OSGI-INF/component.xml
```

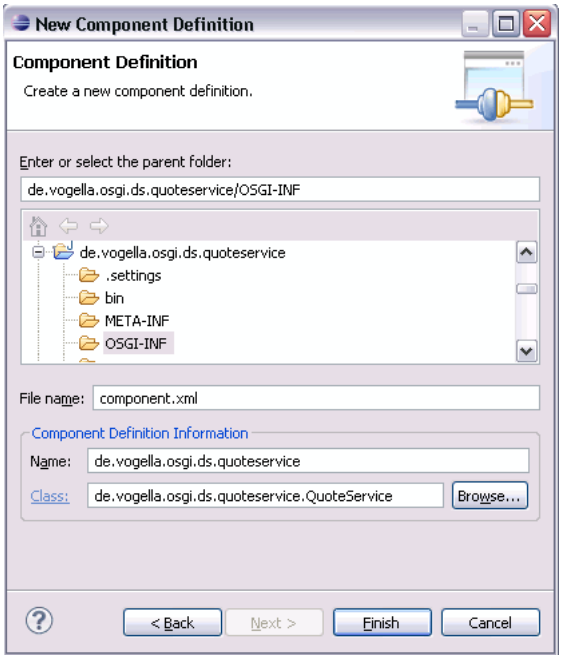
7.2. Define a declarative service

The following will define a DS service based on the quote example. It is therefore required that you have created the "de.vogella.osgi.quote" project which contains the interface definition.

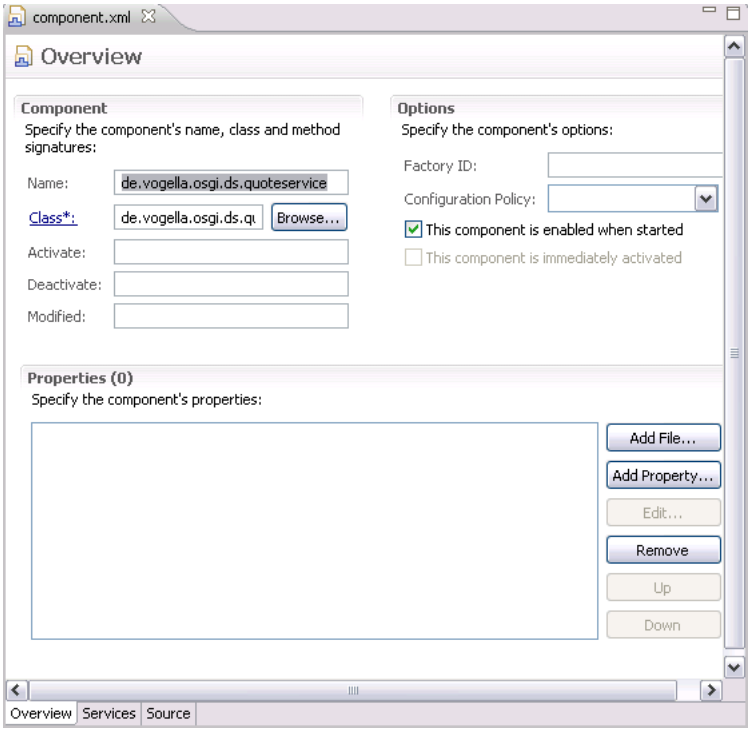
Create a new plugin project "de.vogella.osgi.ds.quoteservice". Do not use a template, do not create an activator. Import package "de.vogella.osgi.quote" in MANIFST.MF on the tab "dependencies".

Create the Folder "OSGI-INF" in your project. Select the new folder, right-click on it and select New -> Other -> Plug-in Development -> Component Definition. The wizard will also add the "Service-Component" entry to "MANIFEST.MF".

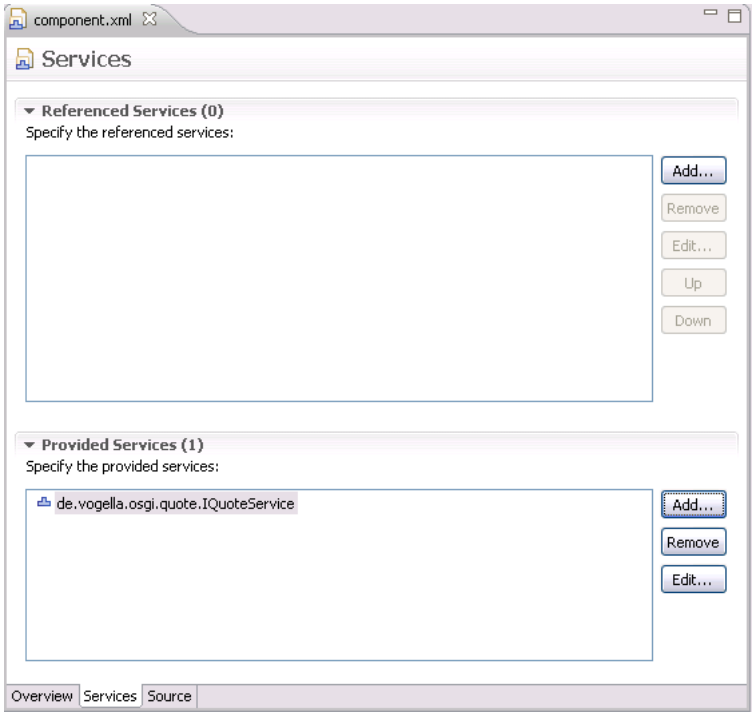




This should open the service editor.



In component.xml switch to the service tab and press "Add" under "Provided Services" and select IQuoteService.



Create now the class "QuoteService" which implements the interface IQuoteService.

```
package de.vogella.osgi.ds.quoteservice;

import java.util.Random;

import de.vogella.osgi.quote.IQuoteService;

public class QuoteService implements IQuoteService {

	@Override
	public String getQuote() {
		Random random = new Random();
		// Create a number between 0 and 2
		int nextInt = random.nextInt(3);
		switch (nextInt) {
		case 0:
			return "Ds: Tell them I said something";
		case 1:
			return "Ds: I feel better already";
		default:
			return "Ds: Hubba Bubba, Baby!";
		}
	}

}
```

You have successfully defined a service via DS.

7.3. Review the result

Go back to component.xml and select the "Source". The declaration of the service looks like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="de.vogella.osgi.ds.quoteservice">
   <implementation class="de.vogella.osgi.ds.quoteservice.QuoteService"/>
   <service>
      <provide interface="de.vogella.osgi.quote.IQuoteService"/>
   </service>
</scr:component>
```

This means that there is a component called "de.vogella.osgi.ds.quoteservice" which provides a service to the OSGI Service Registry under the interface "de.vogella.osgi.quote.IQuoteService", .and the component is implemented by the class "de.vogella.osgi.ds.quoteservice.QuoteService.

Check now the MANIFEST.MF. Here you find the entry: Service-Component: OSGI-INF/component.xml"

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Quoteservice
Bundle-SymbolicName: de.vogella.osgi.ds.quoteservice
Bundle-Version: 1.0.0.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Import-Package: de.vogella.osgi.quote
Service-Component: OSGI-INF/component.xml
```

This defines that the plugin provides this services.

7.4. Run declarative services

Declaratives service require a few more bundles. In addition to "org.eclipse.osgi" you also require.

- org.eclipse.equinox.util
- org.eclipse.equinox.ds - Declarative service

Copy the "org.eclipse.equinox.ds*.jar", "org.eclipse.osgi.services.jar" and "org.eclipse.equinox.util*.jar" from your Eclipse/plugin installation directory into a folder, e.g. "C:\temp\bundles\plugins" and install the bundle into your OSGi runtime via.

```
install file:c:\temp\bundles\plugins\org.eclipse.equinox.ds.jar
install file:c:\temp\bundles\plugins\org.eclipse.equinox.util.jar
install file:c:\temp\bundles\plugins\org.eclipse.osgi.services.jar
```

Export your bundle to "de.vogella.osgi.ds.quoteservice.jar". and install it via:

```
install file:c:\temp\bundles\plugins\de.vogella.osgi.ds.quoteservice.jar
```

To check if your service was registered use the command "services". This will list all installed and available services.

If you stop / uninstall the old service provider and start the new one your service should be picked up by the consumer.

```
osgi> ss

Framework is launched.

id      State       Bundle
0       ACTIVE      org.eclipse.osgi_3.5.1.R35x_v20090827
5       RESOLVED    de.vogella.osgi.firstbundle_1.0.0.200912211101
7       ACTIVE      de.vogella.osgi.quote_1.0.0
9       RESOLVED    de.vogella.osgi.quoteservice_1.0.0
10      ACTIVE      de.vogella.osgi.quoteconsumer_1.0.0
22      ACTIVE      org.eclipse.equinox.ds_1.1.1.R35x_v20090806
23      RESOLVED    de.vogella.osgi.ds.quoteservice_1.0.0.200912221341
26      ACTIVE      org.eclipse.equinox.util_1.0.100.v20090520-1800
27      RESOLVED    org.eclipse.osgi.services_3.2.0.v20090520-1800

osgi> start 23

osgi> Ds: Tell them I said something
Ds: I feel better already
```

8. Using services via declarative services

Of course you can also define the consumption of services via DS.

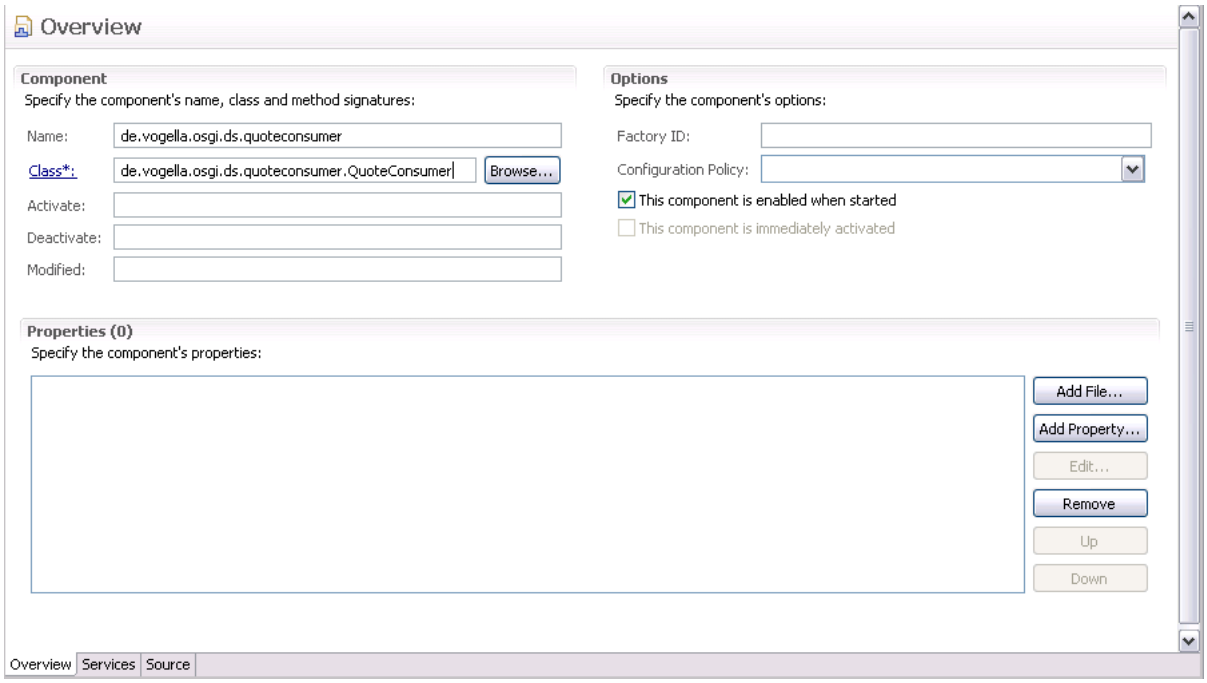
Create a new plugin "de.vogella.osgi.ds.quoteconsumer". Do not use a template, do not create an activator. Import the package "de.vogella.osgi.quote" in MANIFEST.MF on the tab "Dependencies".

Create the following class.

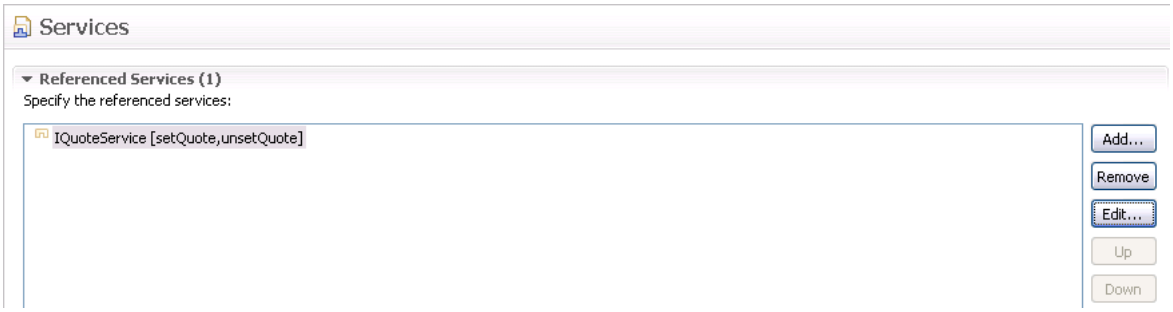
```
package de.vogella.osgi.ds.quoteconsumer;

import de.vogella.osgi.quote.IQuoteService;

public class QuoteConsumer {
	private IQuoteService service;

	public void quote() {
		System.out.println(service.getQuote());
	}

	// Method will be used by DS to set the quote service
	public synchronized void setQuote(IQuoteService service) {
		System.out.println("Service was set. Thank you DS!");
		this.service = service;
		// I know I should not use the service here but just for demonstration
		System.out.println(service.getQuote());
	}

	// Method will be used by DS to unset the quote service
	public synchronized void unsetQuote(IQuoteService service) {
		System.out.println("Service was unset. Why did you do this to me?");
		if (this.service == service) {
			this.service = null;
		}
	}
}
```

Note that this class has no dependency to OSGi.

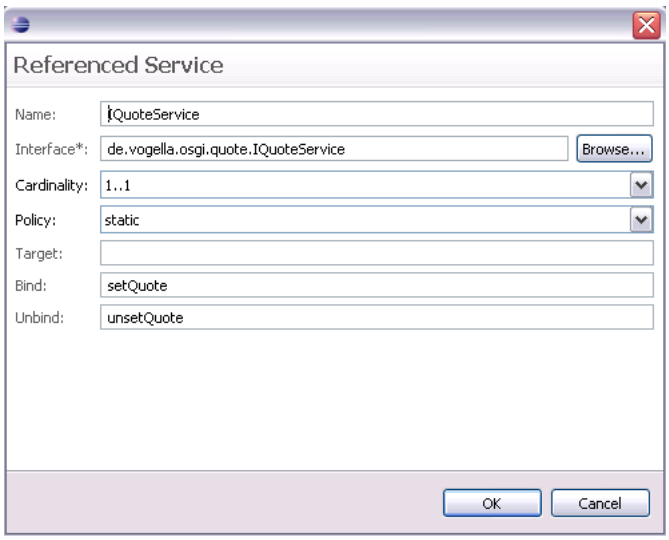
Create the Folder "OSGI-INF" and create a new "Component Definition" in this folder.



This time we will use a service. Maintain the "Referenced Services".



Make the relationship to the bind / unbind method via by selecting your entry can pressing "Edit".



The result component.xml should look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="de.vogella.osgi.ds.quoteconsumer">
   <implementation class="de.vogella.osgi.ds.quoteconsumer.QuoteConsumer"/>
   <reference bind="setQuote" cardinality="1..1" interface="de.vogella.osgi.quote.IQuoteService" name="IQuoteService" policy="static" unbind="unsetQuote"/>
</scr:component>
```

The result MANIFEST.MF should look like:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Quoteconsumer
Bundle-SymbolicName: de.vogella.osgi.ds.quoteconsumer
Bundle-Version: 1.0.4
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Import-Package: de.vogella.osgi.quote
Service-Component: OSGI-INF/component.xml
```

Export your plugin and install it via: install file:c:\temp\bundles\plugins \de.vogella.osgi.ds.quoteconsumer.jar

"If you start the bundle now with "start id_of_your_bundle" you should get the feedback that the service was set and one quote should be returned

9. Thank you

Please help me to support this article:

10. Questions and Discussion

Before posting questions, please see the vogella FAQ . If you have questions or find an error in this article please use the www.vogella.de Google Group . I have created a short list how to create good questions which might also help you.

11. Links and Literature

11.1. Source Code

Source Code of Examples

11.2. OSGi Resources

http://www.osgi.org OSGi Homepage

http://www.eclipse.org/equinox Equinox Homepage

http://www.eclipse.org/equinox/documents/quickstart.php Equinox Quickstart guide

http://www.ibm.com/developerworks/opensource/library/os-osgiblueprint/ OSGi Blueprint services

11.3. vogella Resources

Eclipse RCP Training Join my Eclipse RCP Training to become an RCP Expert in 5 days (Training in German)

Android Tutorial Introduction to Android Programming

GWT Tutorial Program in Java and compile to JavaScript and HTML

Eclipse RCP Tutorial Create native applications in Java

JUnit Tutorial Test your application

Git Tutorial Put everything you have under distributed version control system