OSGi Services - Tutorial

segunda-feira, 25 maio 2015, 11:50 AM

OSGi Services - Tutorial

Lars Vogel

Version 5.1

18.07.2014

OSGi services with Eclipse Equinox

This tutorial explains the usage of OSGi services primary focusing on the usage of declarative services. Eclipse Equinox is used as an stand-alone OSGi server. For this tutorial Eclipse 4.4 (Luna) is used.

Table of Contents

1. Prerequisite

The following assumes that you are familiar with the OSGi runtime and its modularity layer as described in **OSGi modularity**.

2. OSGi Services

2.1. What are OSGi services?

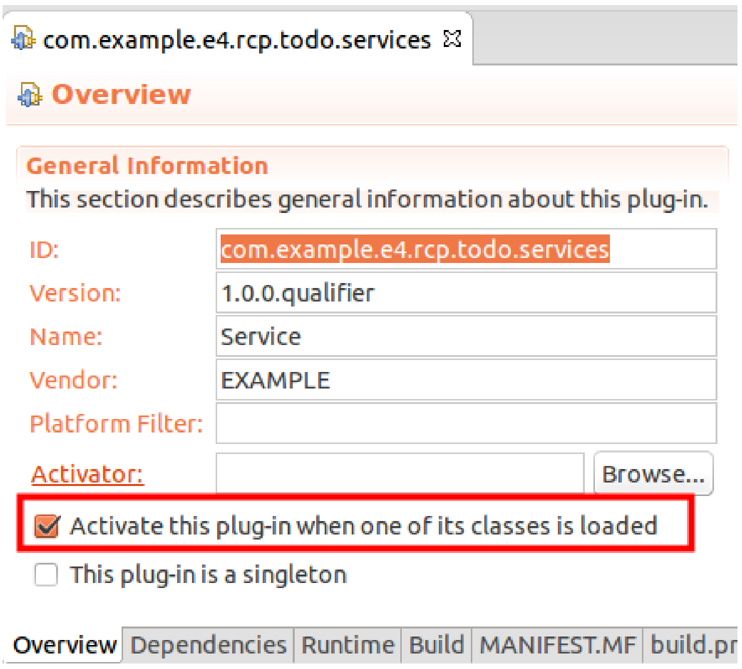
A *service* in OSGi is defined by a standard Java class or interface. A plug-in can register new services and consume existing services via the OSGi runtime. OSGi provides a central *service registry* for this purpose.

A service can be dynamically started and stopped, and plug-ins which use services must be able to handle this dynamic behavior. The plug-ins can register listeners to be informed if a service is started or stopped.

2.2. Life cycle status for providing services

To provide a service a plug-in needs to be in the `ACTIVE` life cycle status of OSGi.

This requires that the service plug-in has the *Activate this plug-in when one of its classes is loaded* flag set in the manifest file.



com.example.e4.rcp.todo.services

Overview

General Information
This section describes general information about this plug-in.

ID:	com.example.e4.rcp.todo.services
Version:	1.0.0.qualifier
Name:	Service
Vendor:	EXAMPLE
Platform Filter:	
Activator:	

[✓] Activate this plug-in when one of its classes is loaded
[ ] This plug-in is a singleton

2.3. Best practices for defining services

It is good practice to define a service via a plug-in which only contains the interface definition. Another plug-in would provide the implementation for this service. This allows you to change the implementation of the service via a different plug-in.

2.4. Service properties

During the declaration of a service it is possible to specify key / values which can be used to configure the service.

2.5. Service priorities

It is possible to define a service ranking for a service via a service property. OSGi assigns by default a value of zero as the service ranking. The higher the ranking the better. Frameworks like the Eclipse dependency injection framework automatically inject the service with the highest service ranking.

The `Constants` class from the `org.osgi.framework` package defines the *`service.ranking`* value via the `Constants.SERVICE_RANKING` constant. This constant can be used to set the integer property of the service ranking.

3. The OSGi declarative services functionality

3.1. Defining declarative services

The OSGi *declarative services* (DS) functionality allows you to define and consume services via metadata (XML) without any dependency in your source code to the OSGi framework.

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 services or by other means.

Service components consist of an 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 it provides. Plug-ins typically define component descriptions in a directory called *`OSGI-INF`*.

A reference to the component description file is entered in the *`MANIFEST.MF`* file via the *`Service-Component`* property. If the OSGi runtime finds such a reference, the `org.eclipse.equinox.ds` plug-in creates the corresponding service.

The following example *`MANIFEST.MF`* file demonstrates how a reference to a component definition file looks like.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Service
Bundle-SymbolicName: com.example.e4.rcp.todo.service
Bundle-Version: 1.0.0.qualifier
Bundle-Vendor: EXAMPLE
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ActivationPolicy: lazy
Service-Component: OSGi-INF/service.xml
```

3.2. Required bundles

To use declarative services the following plug-ins must be available at runtime.

- `org.eclipse.equinox.util` - contains utility classes
- `org.eclipse.equinox.ds` - is responsible for reading the component metadata and for creating and registering the services based this information
- org.eclipse.osgi.services - service functionality used by declarative services

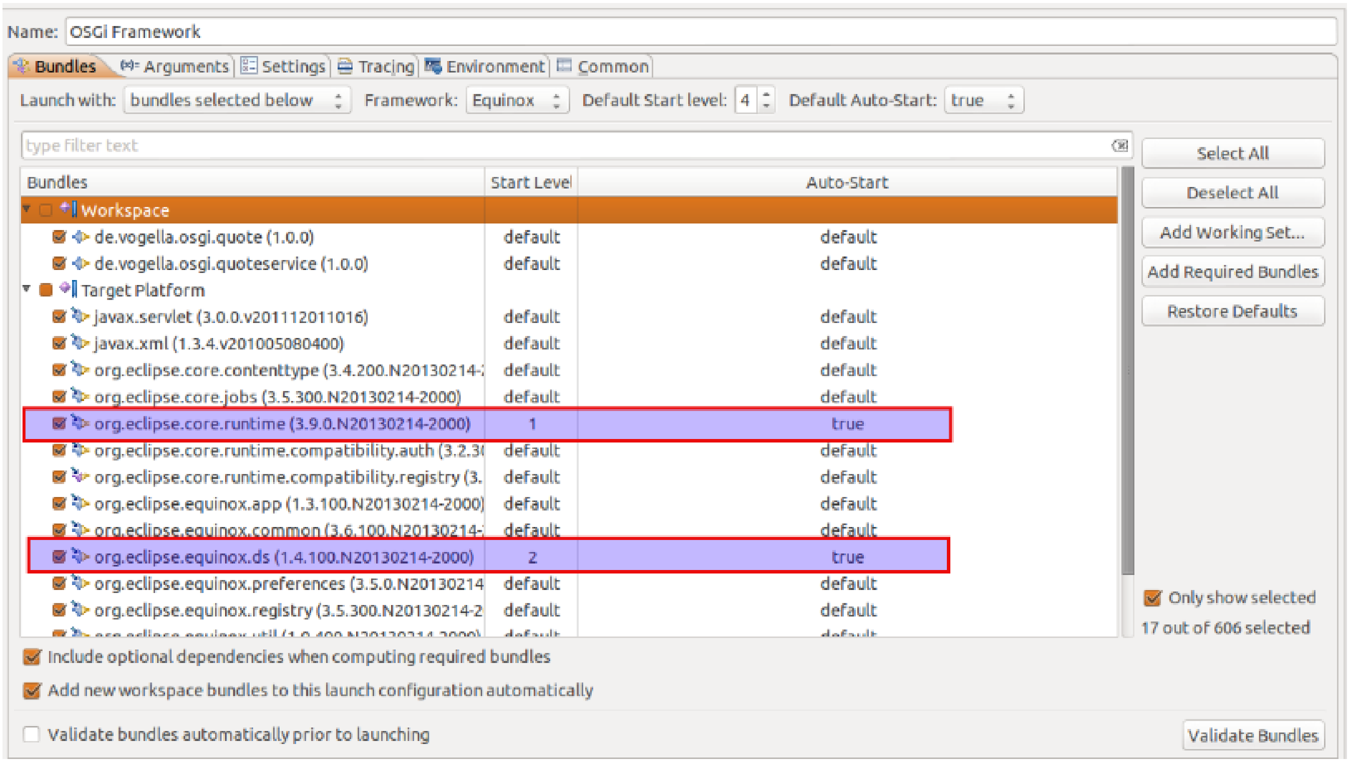
3.3. Activating the declarative service plug-in

A plug-in which provides service must be in its `ACTIVE` life cycle status. Therefore, ensure that the *Activate this plug-in when one of its classes is loaded* flag is set on the *`MANIFEST.MF`* file. See **Section 2.2, "Life cycle status for providing services"** for details.

4. Setting the start level for declarative services

If you use OSGi DS services outside Eclipse RCP applications, you need to ensure that the `org.eclipse.equinox.ds` plug-ins is started before any application plug-in which wants to consume a service.

You can ensure this in your launch configuration by setting the *auto-start* field to true and the start level lower than 4 (4 is the default value) for the org.eclipse.equinox.ds plug-in



Name: OSGi Framework

Launch with: bundles selected below | Framework: Equinox | Default Start level: 4 | Default Auto-Start: true

Bundles	Start Level	Auto-Start
Workspace		
de.vogella.osgi.quote (1.0.0)	default	default
de.vogella.osgi.quoteservice (1.0.0)	default	default
Target Platform		
javax.servlet (3.0.0.v201112011016)	default	default
javax.xml (1.3.4.v201005080400)	default	default
org.eclipse.core.contenttype (3.4.200.N20130214-:	default	default
org.eclipse.core.jobs (3.5.300.N20130214-2000)	default	default
org.eclipse.core.runtime (3.9.0.N20130214-2000)	1	true
org.eclipse.core.runtime.compatibility.auth (3.2.3(	default	default
org.eclipse.core.runtime.compatibility.registry (3.	default	default
org.eclipse.equinox.app (1.3.100.N20130214-2000)	default	default
org.eclipse.equinox.common (3.6.100.N20130214-	default	default
org.eclipse.equinox.ds (1.4.100.N20130214-2000)	2	true
org.eclipse.equinox.preferences (3.5.0.N20130214	default	default
org.eclipse.equinox.registry (3.5.300.N20130214-2	default	default

Only show selected: checked
17 out of 606 selected

Include optional dependencies when computing required bundles: checked
Add new workspace bundles to this launch configuration automatically: checked
Validate bundles automatically prior to launching: unchecked

5. Steps to declare an OSGi service

5.1. Defining the service interface

The first step to define an OSGi service is to define the class or interface for which you want to provide a service. This is called the *service interface*, even though it can also be a Java class.

5.2. Providing a service implementation

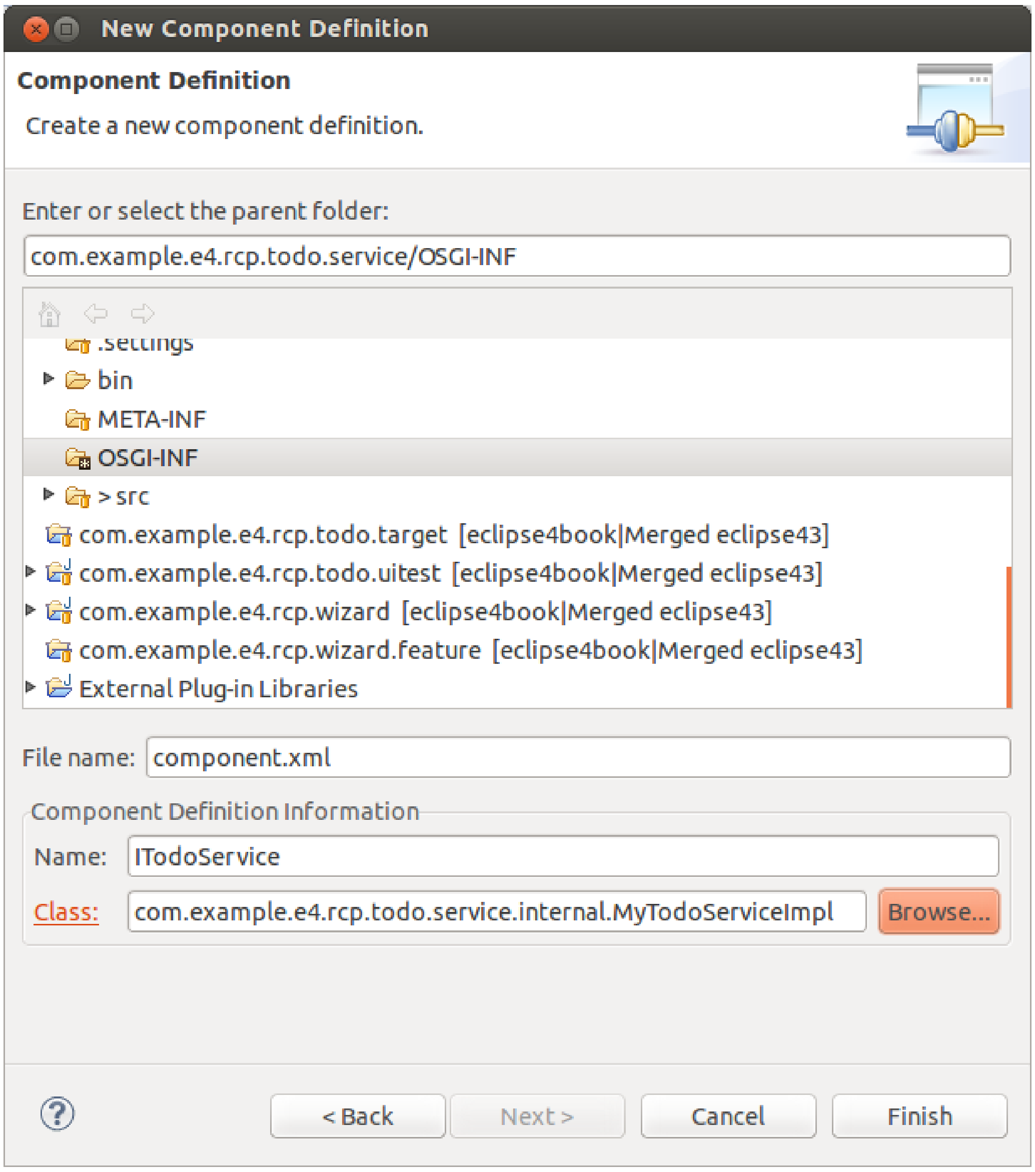
As a second step you write the implementation class for the service interface.

5.3. Service declaration with a component definition file

After you provided the implementation you need to register it for the service interface. In OSGi DS this is done via a component definition file.

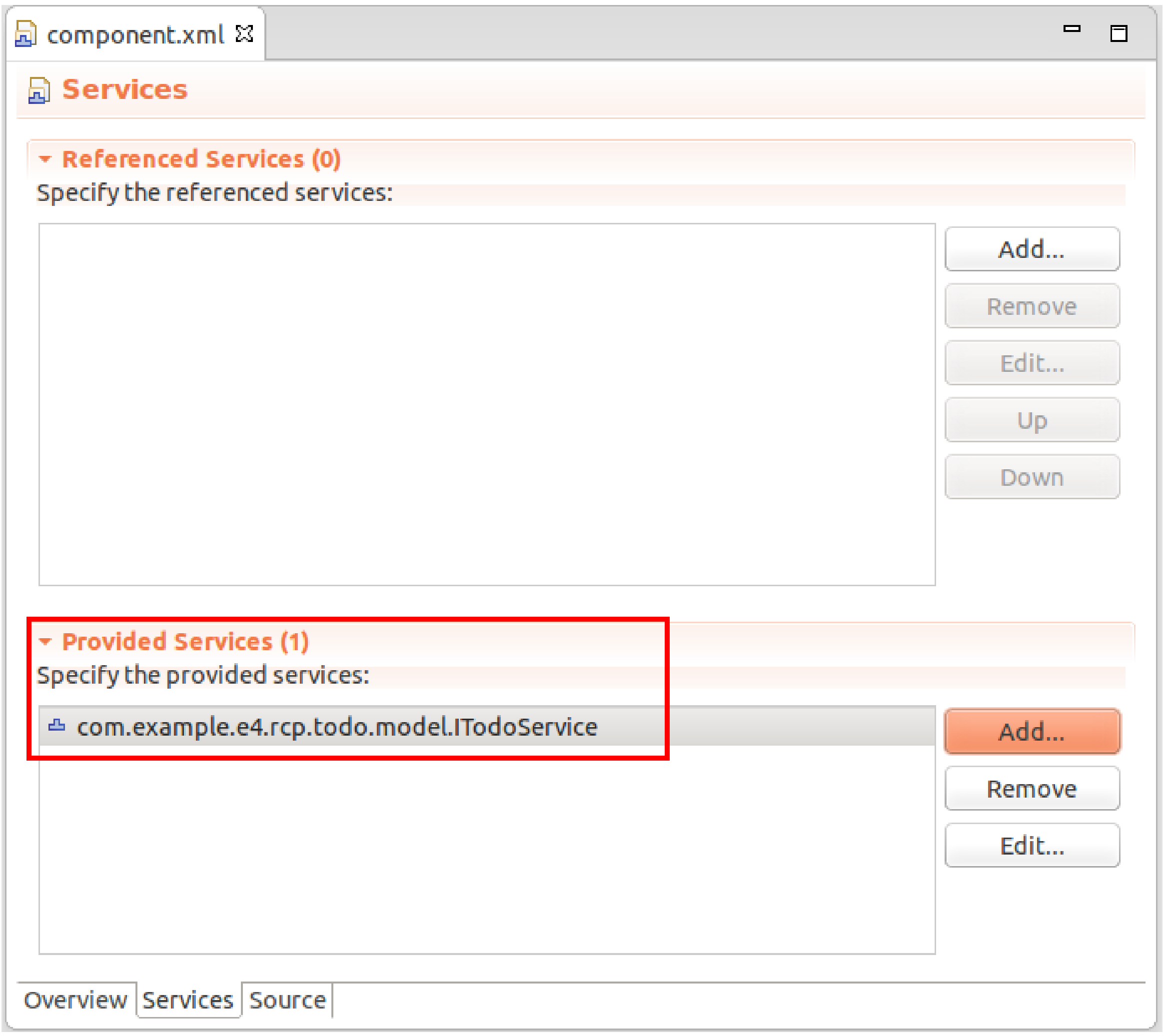
The Eclipse IDE provides a wizard for creating such files via the *New → Other... → Plug-in Development → Component Definition* menu entry. This wizard also adds the `Service-Component` entry to the *`MANIFEST.MF`* file.

On the first page of the wizard, you can enter the filename of the component definition file, a component name and the class which implements the service interface.


New Component Definition
Component Definition
Create a new component definition.
Enter or select the parent folder:
com.example.e4.rcp.todo.service/OSGI-INF
.settings
bin
META-INF
OSGI-INF
> src
com.example.e4.rcp.todo.target [eclipse4book|Merged eclipse43]
com.example.e4.rcp.todo.uitest [eclipse4book|Merged eclipse43]
com.example.e4.rcp.wizard [eclipse4book|Merged eclipse43]
com.example.e4.rcp.wizard.feature [eclipse4book|Merged eclipse43]
External Plug-in Libraries
File name:
component.xml
Component Definition Information
Name:
ITodoService
Class:
com.example.e4.rcp.todo.service.internal.MyTodoServiceImpl
Browse...
< Back
Next >
Cancel
Finish

If you press the *Finish* button, the file is created and the corresponding editor opens.

In this editor you can specify the provided and required service on the *Services* tab. To provide a service you press the *Add...* button under *Provided Services* and select the service interface you want to provide.



For example assume that you want to provide a service for the *ITodoService* interface via the `MyTodoServiceImpl` class. A correctly maintained *component.xml* XML file would look like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="ITodoService">
   <implementation class="com.example.e4.rcp.todo.service.internal.MyTodoServiceImpl"/>
   <service>
      <provide interface="com.example.e4.rcp.todo.model.ITodoService"/>
   </service>
</scr:component>
```

5.4. Reference to the service in the MANIFEST.MF file

After the definition of the component your *MANIFEST.MF* file contains an entry to the service component.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Service
Bundle-SymbolicName: com.example.e4.rcp.todo.service
Bundle-Version: 1.0.0.qualifier
Bundle-Vendor: EXAMPLE
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Require-Bundle: com.example.e4.rcp.todo.model;bundle-version="1.0.0"
```

```
Bundle-ActivationPolicy: lazy
Service-Component: OSGI-INF/component.xml
```

5.5. Low-level OSGi service API

OSGi also provides a low-level API for starting, stopping and tracking services. See **Section 8.1, “Using the service API”** for a reference.

6. Tutorial: Define a declarative OSGi 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 plug-in 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 *OSGI-INF* folder in your project. Create a new component definition as described earlier. The implementing class is de.vogella.osgi.ds.quoteservice.QuoteService which provides the service for IQuoteService.

Create 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!";
    }
  }

}
```

Open component.xml and select the tab "Source". The final result should look like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="ITodoService">
   <implementation class="com.example.e4.rcp.todo.service.internal.MyTodoServiceImpl"/>
   <service>
      <provide interface="com.example.e4.rcp.todo.model.ITodoService"/>
   </service>
```

```
</scr:component>
```

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
```

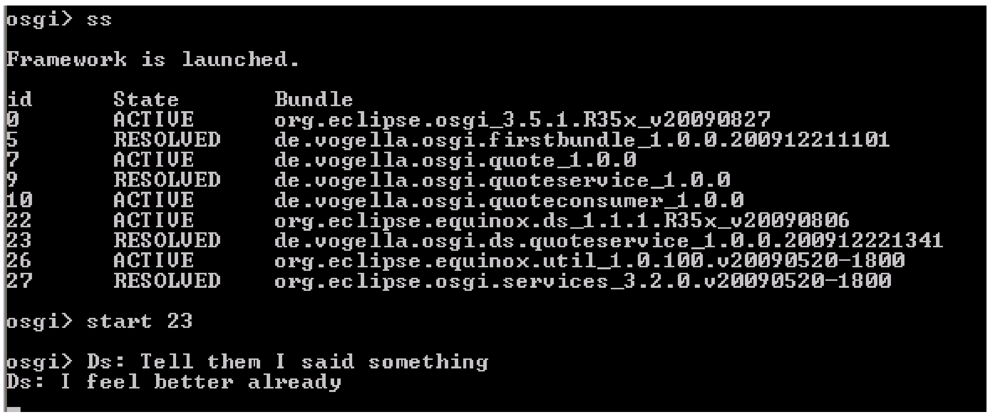
Start the bundles manually so that declarative services are available.

Export your own 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
```

7. Tutorial: Using services via declarative services

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

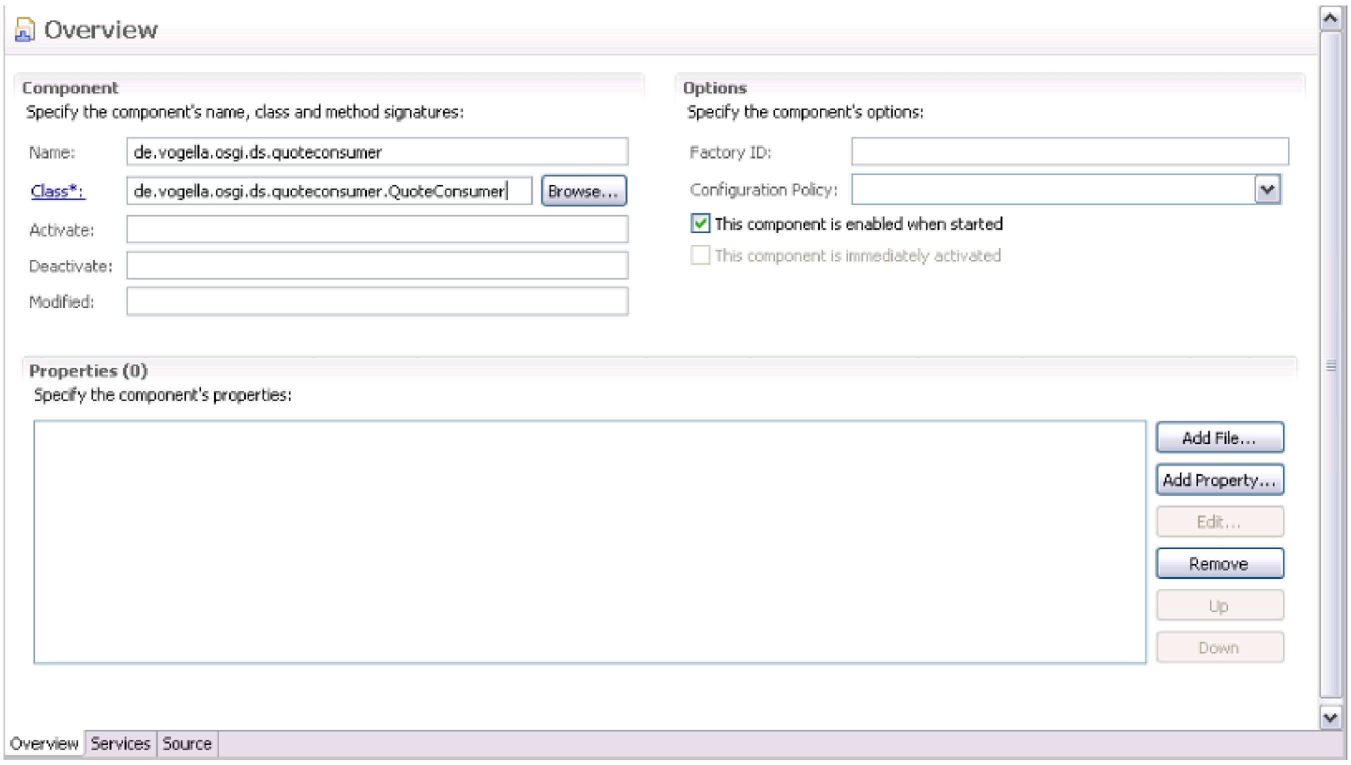
Create a new plug-in "de.vogella.osgi.ds.quoteconsumer". Do not use a template, do not create an activator. Import the package "de.vogella.osgi.quote" in MANIFEST.MF on the *Dependencies* tab.

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;
    }
  }
}
```

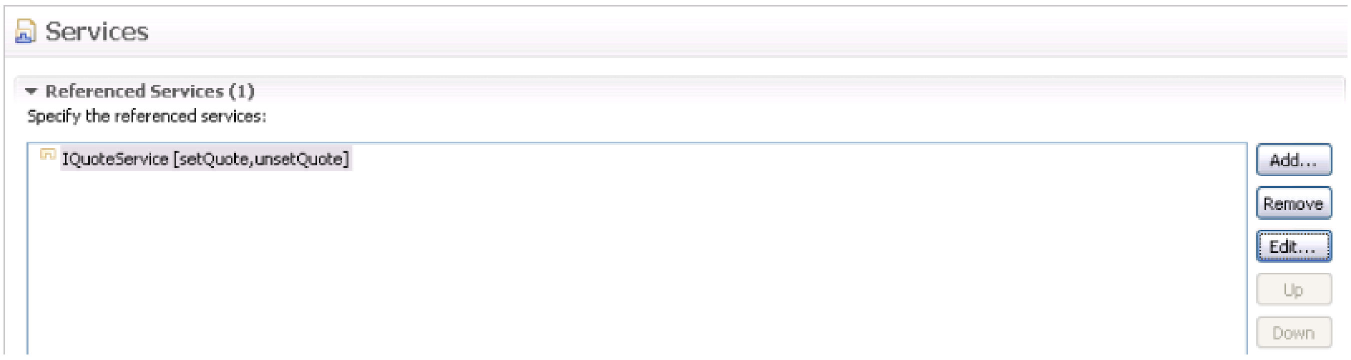
Tip: Note that this class has no dependency to OSGi.

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


Overview
Component
Specify the component's name, class and method signatures:
Options
Specify the component's options:
Properties (0)
Specify the component's properties:
Add File...
Add Property...
Edit...
Remove
Up
Down
Browse...
Overview
Services
Source

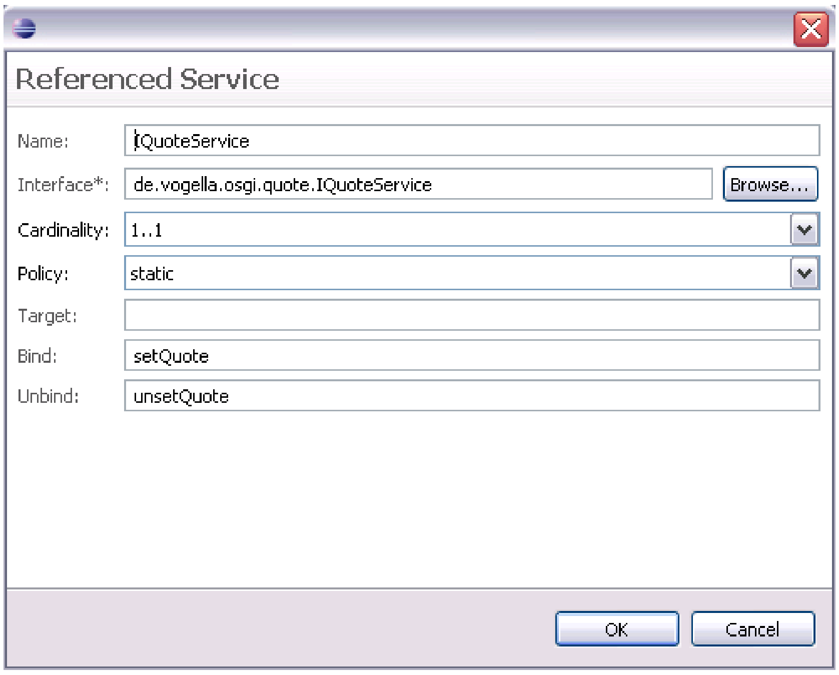
Name: de.vogella.osgi.ds.quoteconsumer
Class*: de.vogella.osgi.ds.quoteconsumer.QuoteConsumer
Activate:
Deactivate:
Modified:
Factory ID:
Configuration Policy:
[✓] This component is enabled when started
[ ] This component is immediately activated

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



IQuoteService [setQuote,unsetQuote]

Make the relationship to the `bind()` and `bind()` method by selecting your entry can by pressing the *Edit* button.



Referenced Service

Name: IQuoteService

Interface*: de.vogella.osgi.quote.IQuoteService

Cardinality: 1..1

Policy: static

Target:

Bind: setQuote

Unbind: unsetQuote

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.IQuoteS
ervice" 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 plug-in and install it via: install file:c:\temp\bundles\plugins \de.vogella.osgi.ds.quoteconsumer.jar

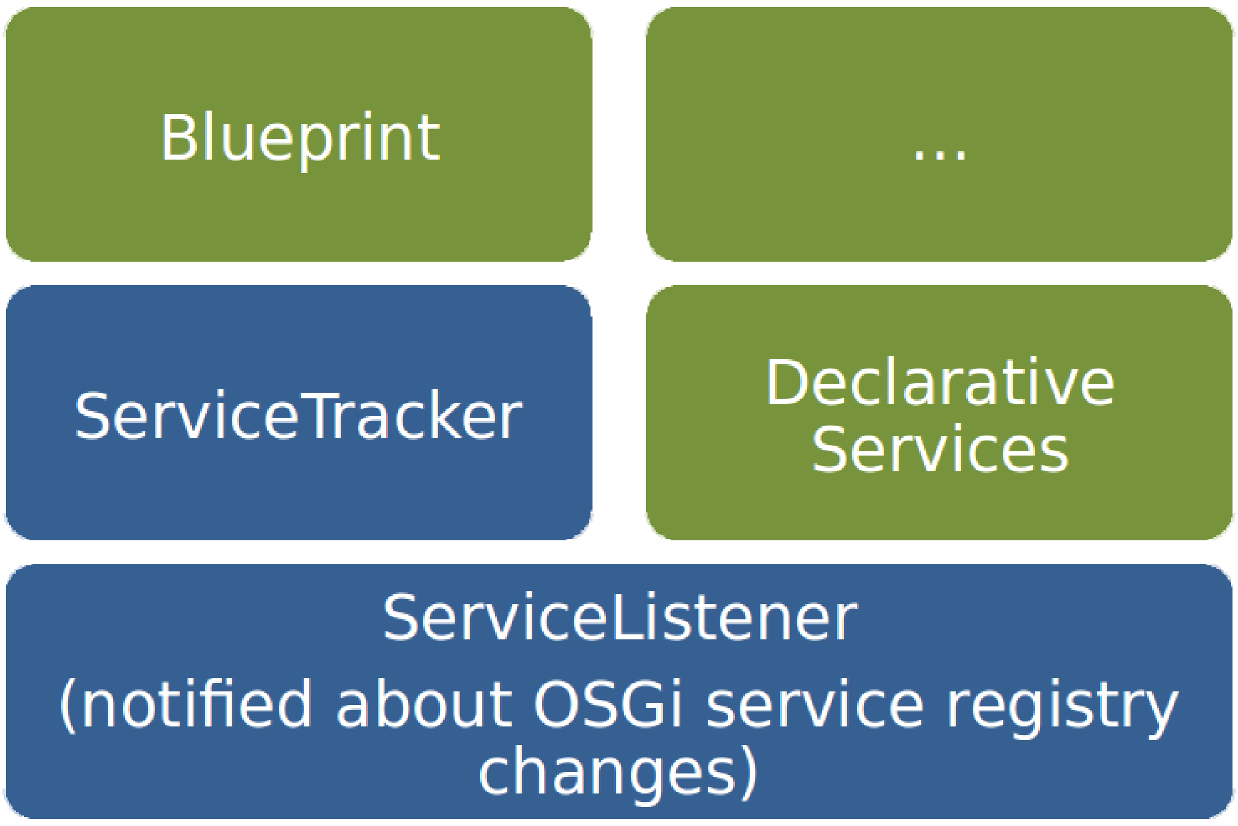
"If you start the bundle now with "start id_of_your_bundle" you should get the feedback that the service was set and one quote should be returned

8. OSGi service low-level API

8.1. Using the service API

OSGi provides several means of declaring services. This book focus on the OSGi declarative service functionality but it is also possible to use other means for defining services. These options are depicted in the following picture. Blueprint and Declarative

Services provide high level abstractions for handling services.


Blueprint
...
ServiceTracker
Declarative Services
ServiceListener (notified about OSGi service registry changes)

This chapter describes the API to work directly with OSGi services but, if you have the option, you should prefer higher level abstractions as these simplify the handling of OSGi services.

8.2. BundleContext

Access to the service registry is performed via the `BundleContext` class.

A bundle can define a `Bundle-Activator` (Activator) in its declaration. This class must implement the `BundleActivator` interface.

If defined, OSGi injects the `BundleContext` into the `start()` and `stop()` methods of the implementing `Activator` class.

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


public class Activator implements BundleActivator {

  public void start(BundleContext context) throws Exception {
    System.out.println("Starting bundle");
    // do something with the context, e.g.
    // register services
  }

  public void stop(BundleContext context) throws Exception {
    System.out.println("Stopping bundle");
    // do something with the context, e.g.
    // unregister service
  }
}
```

If you do not have an `Activator`, you can use the `FrameworkUtil` class from the OSGi framework which allows you to retrieve the `BundleContext` for a class.

```
BundleContext bundleContext =
        FrameworkUtil.
        getBundle(this.getClass()).
        getBundleContext();
```

8.3. Registering services via API

A bundle can also register itself for the events (`ServiceEvents`) of the `BundleContext`. These are, for example, triggered if a new bundle is installed or de-installed or if a new service is registered.

To publish a service in your bundle use:

```
public class Activator implements BundleActivator {
  // ...
  public void start(BundleContext context) throws Exception {
    context.
      registerService(IMyService.class.getName(),
         new ServiceImpl(), null);

  }
  // ...
}
```

Once the service is no longer used, you must unregister the service with OSGi. OSGi counts the usage of services to enable the dynamic replacement of services. So once the service is no longer used by your implementation, you should tell the OSGi environment this by:

```
context.ungetService(serviceReference);
```

In the `registerService()` method from the `BundleContext` class you can specify arbitrary properties in the dictionary parameter.

You can use the `getProperty()` method of the `ServiceReference` class from the `org.osgi.framework` package, to access a specific property.

8.4. Accessing a service via API

A bundle can acquire a service via the `BundleContext` class. The following example demonstrates that.

```
ServiceReference<?> serviceReference = context.
  getServiceReference(IMyService.class.getName());
IMyService service = (IMyService) context.
  getService(serviceReference);
```

8.5. Low-level API vs OSGi declarative services

OSGi services can be dynamically started and stopped. If you work with the OSGI low-level API you have to handle this dynamic in your code. This make the source code unnecessary complex. If you do not handle that correctly your service consumer can keep a reference to the service and the service cannot be removed via the OSGi framework.

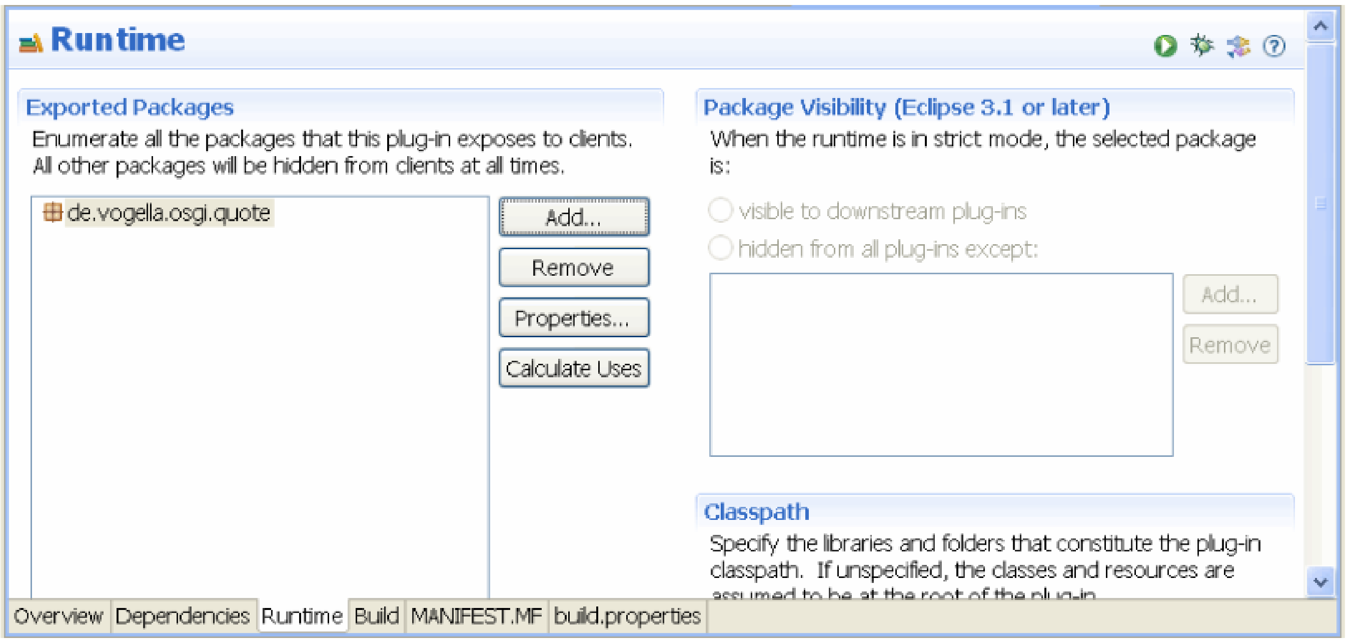
To handle the dynamics automatically declarative services were developed. Prefer therefore the usage of OSGi *declarative services* over the low-level API.

9. Tutorial: Using the OSGi service API

In the following we will define and consume a service. Our service will return "famous quotes".

9.1. Define the service interface

Create a plug-in 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.


Runtime
Exported Packages
Enumerate all the packages that this plug-in exposes to clients.
All other packages will be hidden from clients at all times.
de.vogella.osgi.quote
Add...
Remove
Properties...
Calculate Uses
Package Visibility (Eclipse 3.1 or later)
When the runtime is in strict mode, the selected package is:
visible to downstream plug-ins
hidden from all plug-ins except:
Add...
Remove
Classpath
Specify the libraries and folders that constitute the plug-in classpath. If unspecified, the classes and resources are
Overview
Dependencies
Runtime
Build
MANIFEST.MF
build.properties

Create the following interface "IQuoteService".

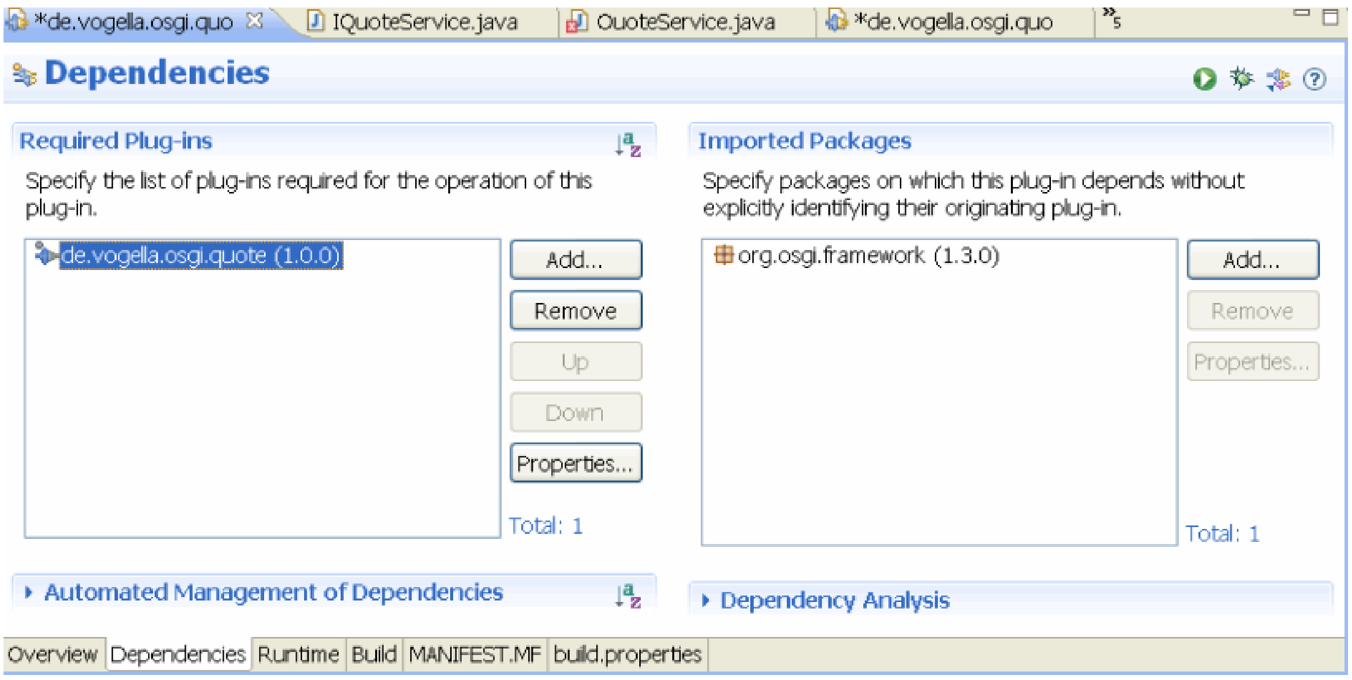
```
package de.vogella.osgi.quote;

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

9.2. Create service

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

Create a plug-in 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".

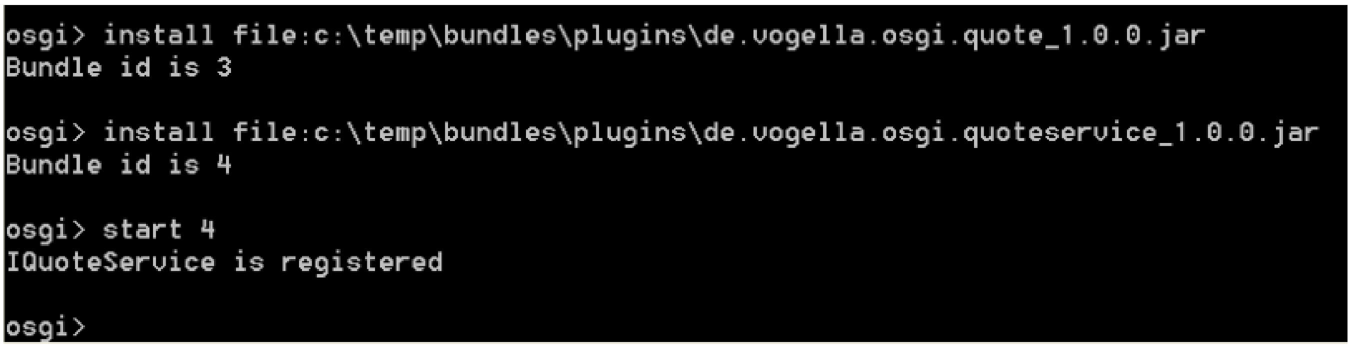
```
package de.vogella.osgi.quoteservice.internal;

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 {
  }
}
```

9.3. Install service bundles

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

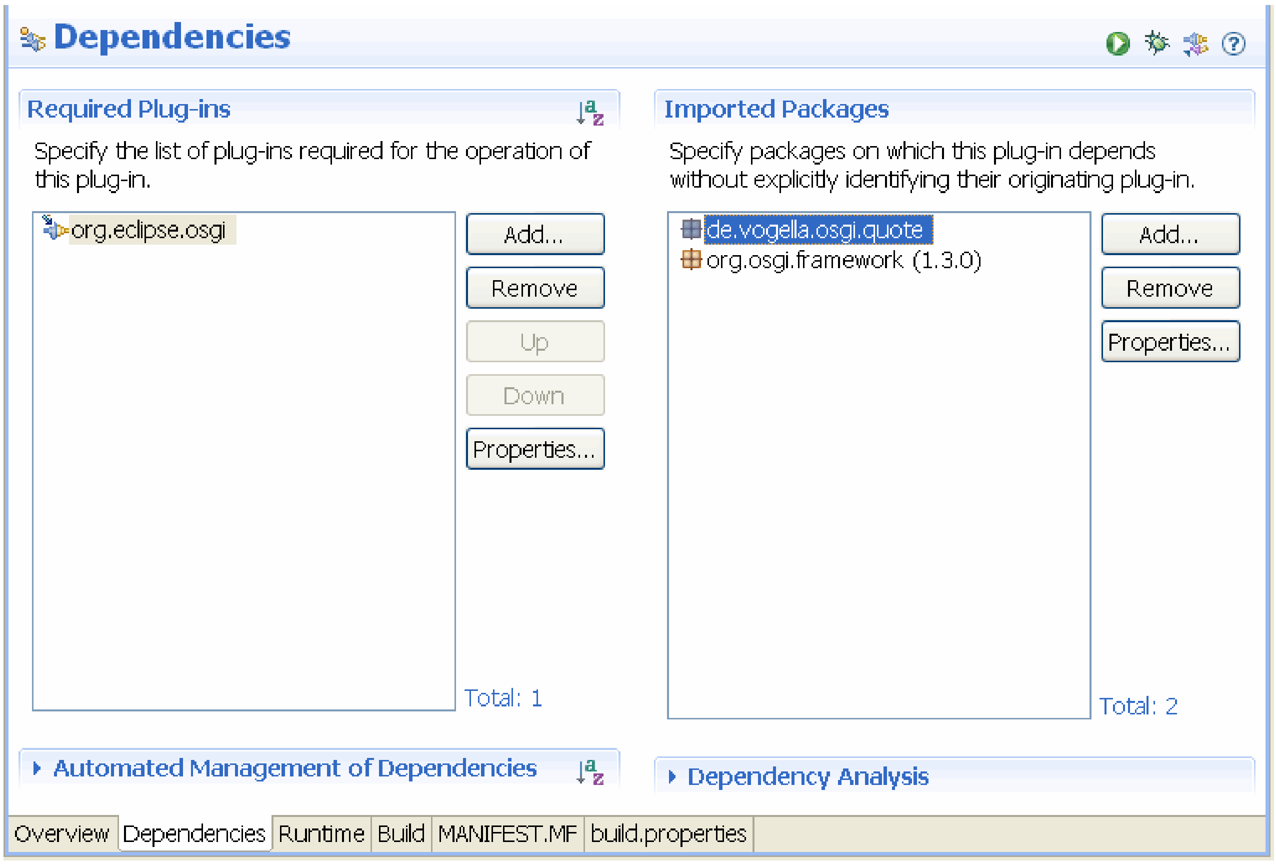


```
osgi> install file:c:\temp\bundles\plugins\de.vogella.osgi.quote_1.0.0.jar
Bundle id is 3

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>
```

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

9.4. Use your service

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


Dependencies

Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in.

org.eclipse.osgi

Total: 1

Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

de.vogella.osgi.quote
org.osgi.framework (1.3.0)

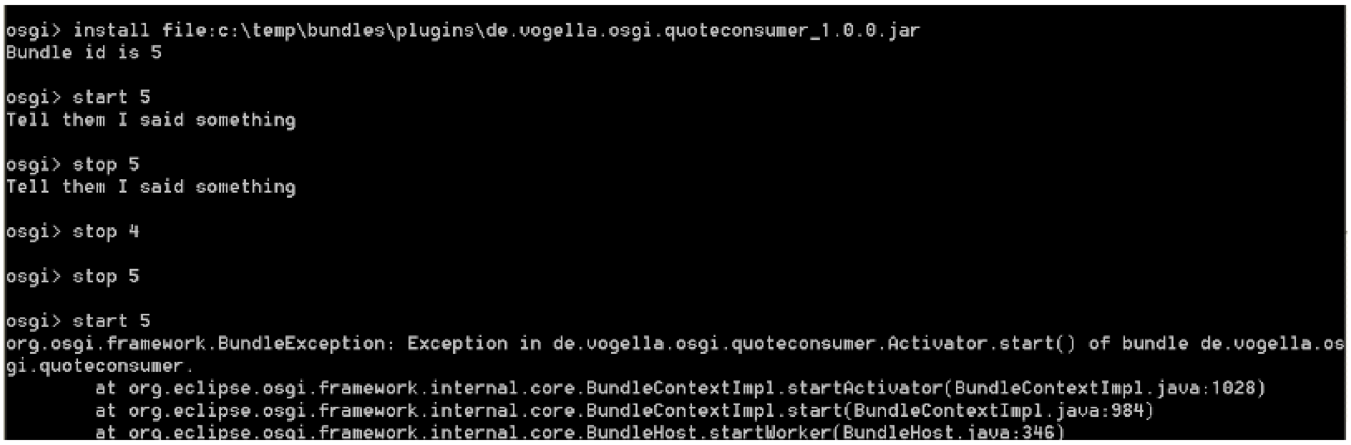
Total: 2

Tip: 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 de.vogella.os
gi.quoteconsumer.
        at org.eclipse.osgi.framework.internal.core.BundleContextImpl.startActivator(BundleContextImpl.java:1028)
        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.

9.5. Use your service with a service tracker

Declare a package dependency to the package "org.osgi.util.tracker" in your bundle.

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();
```

```java
    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.

```java
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(contex
t);
    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.

10. Bndtools

Eclipse use the PDE tooling to manage bundles. Alternatively you can use Bndtools hosted at **http://bndtools.org/**.

Please see **Bndtools tutorial** for an introduction.

11. About this website

11.1. Donate to support free tutorials

Please consider a contribution if this article helped you. It will help to maintain our content and our Open Source activities.

11.2. Questions and discussion

Writing and updating these tutorials is a lot of work. If this free community service was helpful, you can support the cause by giving a tip as well as reporting typos and factual errors.

If you find errors in this tutorial, please notify me (see the **top of the page**). Please note that due to the high volume of feedback I receive, I cannot answer questions to your implementation. Ensure you have read the **vogella FAQ** as I don't respond to questions already answered there.

11.3. License for this tutorial and its code

12. Links and Literature

12.1. Source Code

Source Code of Examples

12.2. OSGi Resources

http://www.osgi.org OSGi Homepage

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

OSGi remote RESTFul service tutorial Equinox Homepage

OSGi remove service tutorial with ECF Equinox 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

12.3. vogella Resources