JBoss OSGi - User Guide

Thomas Diesler <thomas.diesler@jboss.org>
David Bosschaert <david@redhat.com>

JBoss OSGi - User Guide

by Thomas Diesler and David Bosschaert

Version: 1.1.0

Published Date: 07-Mar-2012

Abstract

The JBoss OSGi User Guide is the starting point for OSGi application development and integration in AS7.

Table of Contents

List of Figures

Chapter 1. Introduction

Content

What is OSGi

The OSGi specifications [http://www2.osgi.org/Release4/HomePage] define a standardized, component-oriented, computing environment for networked services that is the foundation of an enhanced service-oriented architecture.

Developing on the OSGi platform means first creating your OSGi bundles, then deploying them in an OSGi Framework.

What does OSGi offer to Java developers?

OSGi modules provide classloader semantics to partially expose code that can then be consumed by other modules. The implementation details of a module, although scoped public by the Java programming language, remain private to the module. On top of that you can install multiple versions of the same code and resolve dependencies by version and other criteria. OSGi also offers advanced lifecycle and services layers, which are explained in more detail further down.

What kind of applications benefit from OSGi?

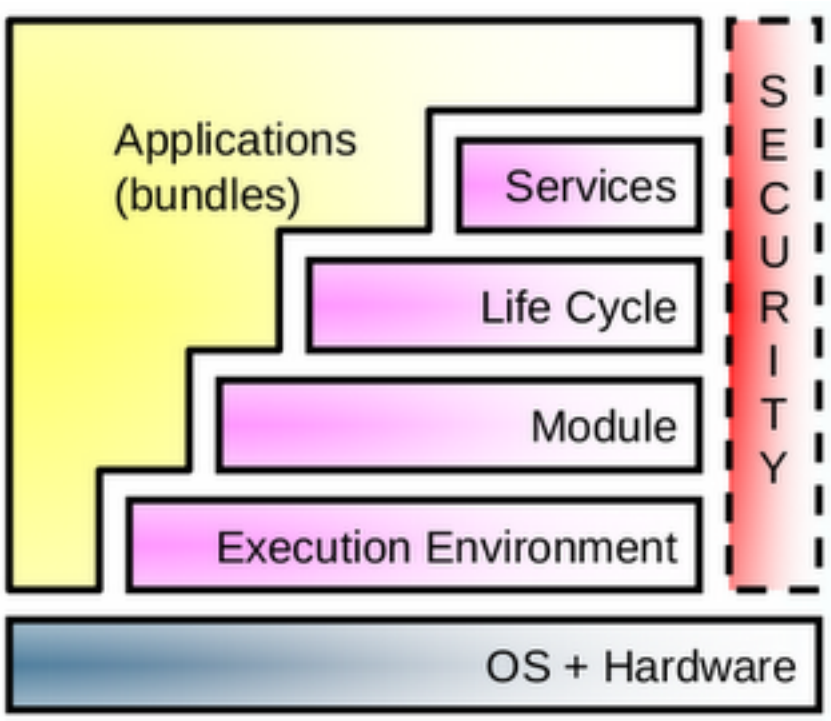
Any application that is designed in a modular fashion where it is necessary to start, stop, update individual modules with minimal impact on other modules. Modules can define their own transitive dependencies without the need to resolve these dependencies at the container level.

OSGi Framework Overview

The functionality of the Framework is divided in the following layers:

- Security Layer (optional)
- Module Layer
- Life Cycle Layer
- Service Layer
- Actual Services

Figure 1.1. Source: OSGi Alliance



OSGi Security Layer

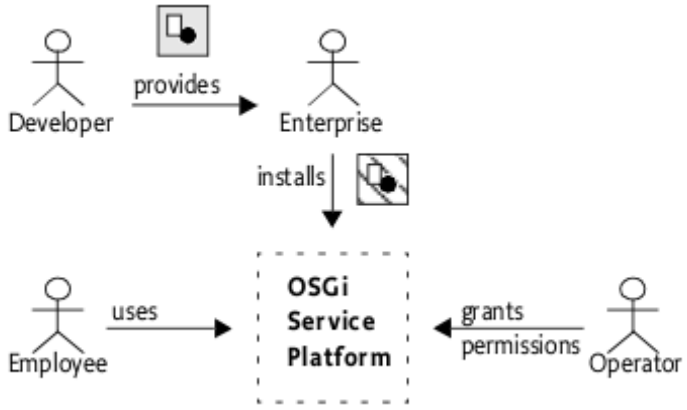
The OSGi Security Layer is an optional layer that underlies the OSGi Service Platform. The layer is based on the Java 2 security architecture. It provides the infrastructure to deploy and manage applications that must run in fine grained controlled environments.

The OSGi Service Platform can authenticate code in the following ways:

- By location
- By signer

For example, an Operator can grant the ACME company the right to use networking on their devices. The ACME company can then use networking in every bundle they digitally sign and deploy on the Operator's device. Also, a specific bundle can be granted permission to only manage the life cycle of bundles that are signed by the ACME company.

Figure 1.2. Source: OSGi Alliance



The current version of JBoss OSGi does not provide this optional layer. If you would like to see this implemented, let us know on the forums: http://community.jboss.org/en/jbossosgi .

OSGi Module Layer

The OSGi Module Layer provides a generic and standardized solution for Java modularization. The Framework defines a unit of modularization, called a bundle. A bundle is comprised of Java classes and other resources, which together can provide functions to end users. Bundles can share Java packages among an exporter bundle and an importer bundle in a well-defined way.

Once a Bundle is started, its functionality is provided and services are exposed to other bundles installed in the OSGi Service Platform. A bundle carries descriptive information about itself in the manifest file that is contained in its JAR file. Here are a few important Manifest Headers defined by the OSGi Framework:

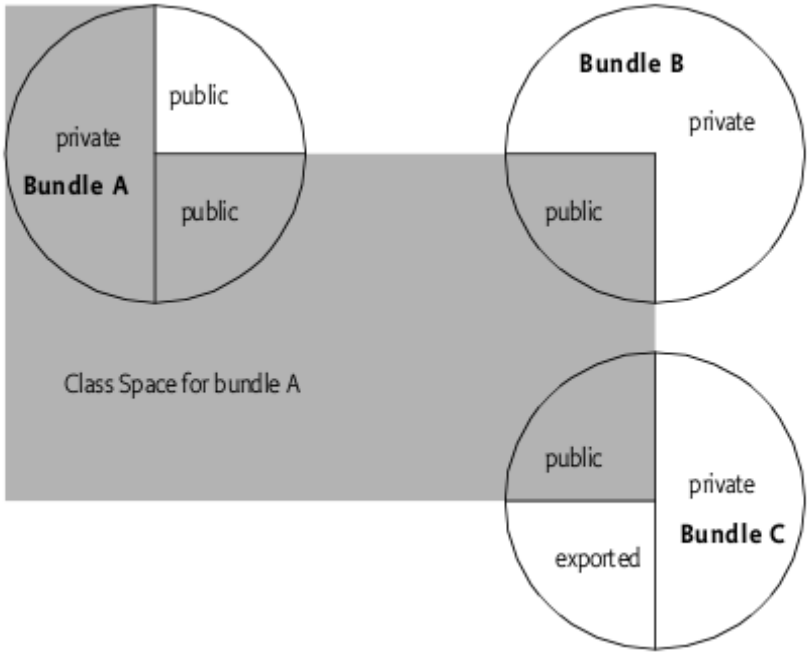
- **Bundle-Activator** - class used to start, stop the bundle
- **Bundle-SymbolicName** - identifies the bundle
- **Bundle-Version** - specifies the version of the bundle
- **Export-Package** - declaration of exported packages
- **Import-Package** - declaration of imported packages

The notion of OSGi Version Range describes a range of versions using a mathematical interval notation. For example

```
Import-Package: com.acme.foo;version="[1.23, 2)", com.acme.bar;version="[4.0, 5.0)
```

With the OSGi Class Loading Architecture many bundles can share a single virtual machine (VM). Within this VM, bundles can hide packages and classes from other bundles, as well as share packages with other bundles.

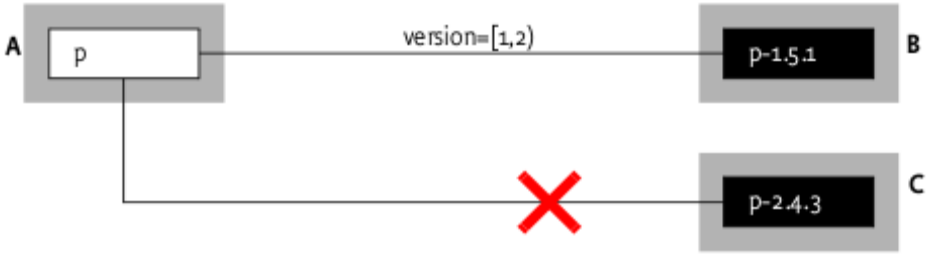
Figure 1.3. Source: OSGi Alliance



For example, the following import and export definition resolve correctly because the version range in the import definition matches the version in the export definition:

```
A: Import-Package: p; version="[1,2)"
    B: Export-Package: p; version=1.5.1
```

Figure 1.4. Source: OSGi Alliance



Apart from bundle versions, OSGi Attribute Matching is a generic mechanism to allow the importer and exporter to influence the matching process in a declarative way. For example, the following statements will match.

```
A: Import-Package: com.acme.foo;company=ACME
    B: Export-Package: com.acme.foo;company=ACME; security=false
```

An exporter can limit the visibility of the classes in a package with the include and exclude directives on the export definition.

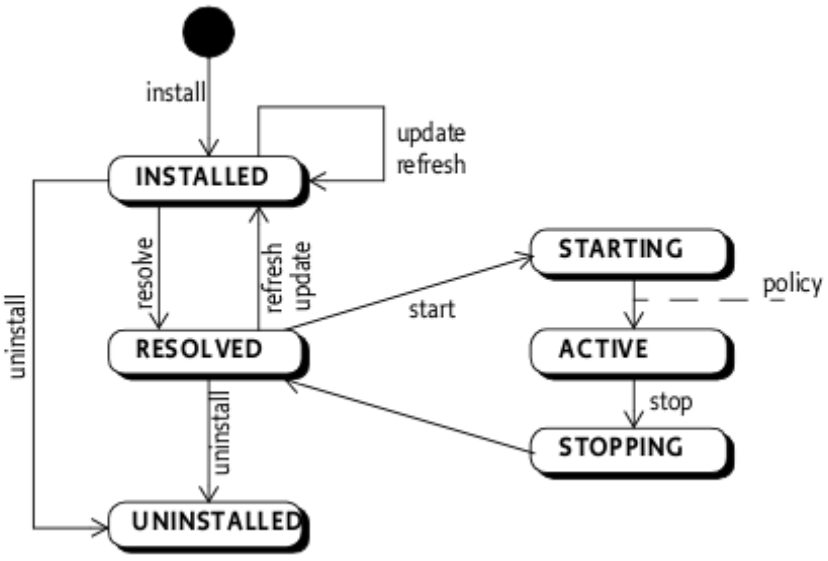
```
Export-Package: com.acme.foo; include:="Qux*,BarImpl"; exclude:=QuxImpl
```

OSGi Life Cycle Layer

The Life Cycle Layer provides an API to control the security and life cycle operations of bundles.

A bundle can be in one of the following states:

Figure 1.5. Source: OSGi Alliance



A bundle is activated by calling its Bundle Activator object, if one exists. The BundleActivator interface defines methods that the Framework invokes when it starts and stops the bundle.

A Bundle Context object represents the execution context of a single bundle within the OSGi Service Platform, and acts as a proxy to the underlying Framework. A *Bundle Context* object is created by the Framework when a bundle is started. The bundle can use this private BundleContext object for the following purposes:

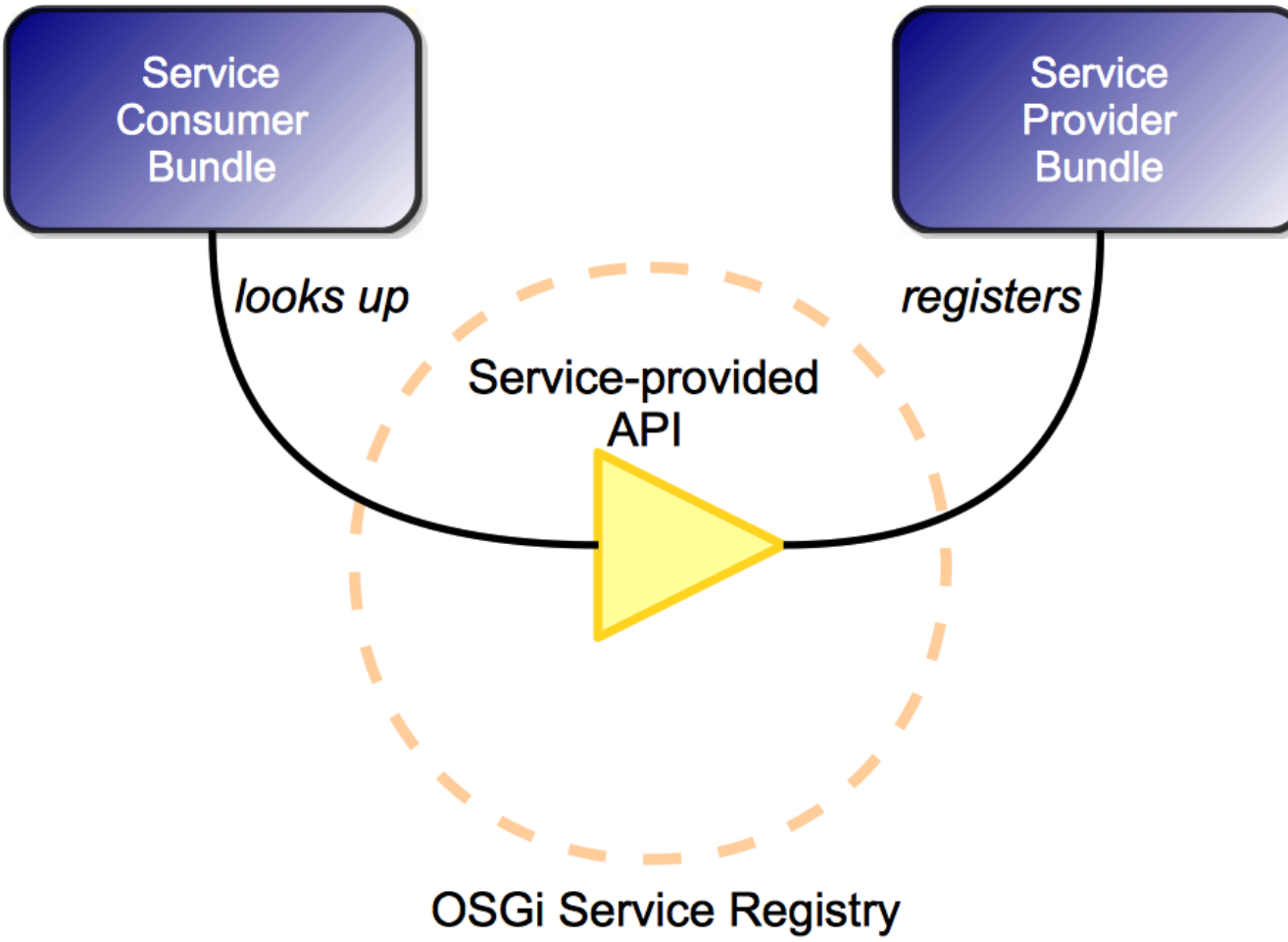
- Installing new bundles into the OSGi environment
- Interrogating other bundles installed in the OSGi environment
- Obtaining a persistent storage area
- Retrieving service objects of registered services
- Registering services in the Framework service
- Subscribing or unsubscribing to Framework events

OSGi Service Layer

The OSGi Service Layer defines a dynamic collaborative model that is highly integrated with the Life Cycle Layer. The service model is a publish, find and bind model. A service is a normal Java object that is registered under one or more Java interfaces with the service registry. OSGi services are dynamic, they can come and go at any time. OSGi service consumers, when written correctly, can deal with this dynamicity. This means that OSGi services provide the capability to create a highly adaptive application which, when written using services, can even be updated at runtime without taking the service consumers down.

The OSGi Declarative Services[8] and OSGi Blueprint [8] specifications significantly simplify the use of OSGi Services which means that a consumer gets the benefits of a dynamic services model for very little effort.

Figure 1.6. OSGi Services



OSGi Service Compendium

The OSGi Service Compendium is described in the OSGi Compendium and Enterprise specifications [http://www.osgi.org/Specifications/HomePage] . It specifies a number of services that may be available in an OSGi runtime environment. Although the OSGi Core Framework specification is useful in itself already, it only defines the OSGi core infrastructure. The services defined in the compendium specification define the scope and functionality of some common services that bundle developers might want to use. Here is a quick summary of the popular ones:

Log Service

Chapter 101 in the Compendium and Enterprise specifications.

The Log Service provides a general purpose message logger for the OSGi Service Platform. It consists of two services, one for logging information and another for retrieving current or previously recorded log information.

The JBoss OSGi Framework provides an implementation of the Log Service which channels logging information through to the currently configured system logger.

Http Service

Chapter 102 in the Compendium and Enterprise specifications.

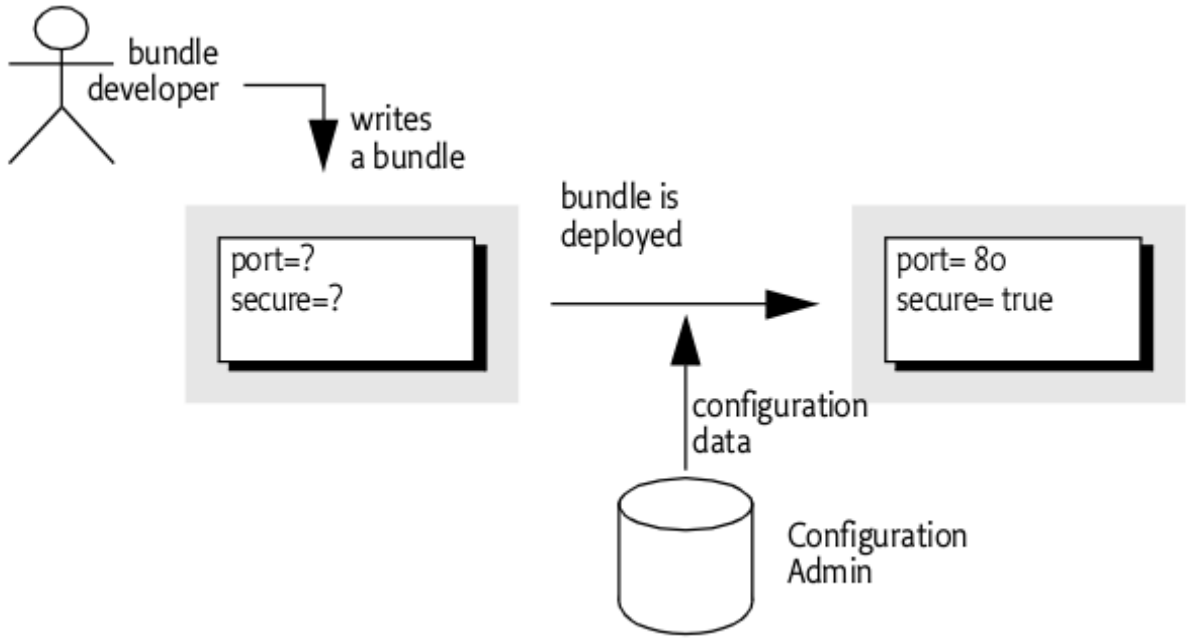
The Http Service supports a standard mechanism for registering servlets and resources from inside an OSGi Framework. This can be used to develop communication and user interface solutions for standard technologies such as HTTP, HTML, XML, etc.

Configuration Admin Service

Chapter 104 in the Compendium and Enterprise specifications.

The Configuration Admin service allows an operator to set the configuration information of deployed bundles.

Figure 1.7. Source: OSGi Alliance



The JBoss OSGi Framework provides an implementation of the Configuration Admin Service which obtains its configuration information from the JBoss Application Server configuration data, for instance the `standalone.xml` file.

Metatype Service

Chapter 105 in the Compendium and Enterprise specifications.

The Metatype Service specification defines interfaces that allow bundle developers to describe attribute types in a computer readable form using so-called metadata. This service is mostly used to define the attributes and datatypes used by Configuration Admin Service information.

User Admin Service

Chapter 107 in the Compendium and Enterprise specifications.

Bundles can use the User Admin Service to authenticate an initiator and represent this authentication as an Authorization object. Bundles that execute actions on behalf of this user can use the Authorization object to verify if that user is authorized.

Declarative Services Specification

Chapter 112 in the Compendium and Enterprise specifications.

The Declarative Services (DS) specification describes a component model to be used with OSGi services. It enables the creation and consumption of OSGi services without directly using any OSGi APIs. Service consumers are informed of their services through injection. The handling of the OSGi service dynamics is done by DS. See also the Blueprint Specification [8] .

Event Admin Service

Chapter 113 in the Compendium and Enterprise specifications.

The Event Admin Service provides an asynchronous inter-bundle communication mechanism. It is based on a event publish and subscribe model, popular in many message based systems.

Blueprint Specification

Chapter 121 in the Enterprise specification.

The OSGi Blueprint Specification describes a component framework which simplifies working with OSGi services significantly. To a certain extent, Blueprint and DS [8] have goals in common, but the realization is different. One of the main differences between Blueprint and DS is in the way service-consumer components react to a change in the availability of required services. In the case of DS the service-consumer will disappear when its required dependencies disappear, while in Blueprint the component stays around and waits for a replacement service to appear. Each model has its uses and it can be safely said that both Blueprint as well as DS each have their supporters. The Blueprint specification was heavily influenced by the Spring framework.

Remote Services Specifications

Chapters 13 and 122 in the Enterprise specification.

OSGi Remote Services add distributed computing to the OSGi service programming model. Where in an ordinary OSGi Framework services are strictly local to the Java VM, with Remote Services the services can be remote. Services are registered and looked up just like local OSGi services, the Remote Services specifications define standard service properties to indicate that a service is suitable for remoting and to find out whether a service reference is a local one or a remote one.

JTA Specification

Chapter 123 in the Enterprise specification.

The OSGi-JTA specification describes how JTA can be used from an OSGi environment. It includes standard JTA-related services that can be obtained from the OSGi registry if an OSGi application needs to make use of JTA.

JMX Specification

Chapter 124 in the Enterprise specification.

The OSGi-JMX specification defines a number of MBeans that provide management and control over the OSGi Framework.

JDBC Specification

Chapter 125 in the Enterprise specification.

The OSGi-JDBC specification makes using JDBC drivers from within OSGi easy. Rather than loading a database driver by class-name (the traditional approach, which causes issues with modularity in general and often requires external access to internal implementation classes), this specification registers the available JDBC drivers under a standard interface in the Service Registry from where they can be obtained by other Bundles without the need to expose internal implementation packages of the drivers.

JNDI Specification

Chapter 126 in the Enterprise specification.

The OSGi-JNDI specification provides access to JNDI through the OSGi Service Registry. Additionally, it provides access to the OSGi Service Registry through JNDI. The special `osgi:` namespace can be used to look up OSGi services via JNDI.

JPA Specification

Chapter 127 in the Enterprise specification.

The OSGi-JPA specification describes how JPA works from within an OSGi framework.

Web Applications Specification

Chapter 128 in the Enterprise specification.

The Web Applications specification describes Web Application Bundles. A WAB is a `.WAR` file which is effectively turned into a bundle. The specification describes how Servlets can interact with the OSGi Service Registry and also how to find all the available Web Applications in an OSGi Framework.

Additionally, the Web Applications spec defines a mechanism to automatically turn an ordinary `.WAR` file into a Web Application Bundle.

Service Tracker Specification

Chapter 701 in the Compendium and Enterprise specifications.

The Service Tracker specification defines a utility class, ServiceTracker. The ServiceTracker API makes tracking the registration, modification, and unregistration of services much easier.

Images courtesy of the OSGi Alliance [http://www.osgi.org] .

Chapter 2. Getting Started

Content

This chapter takes you through the first steps of getting JBoss OSGi and provides the initial pointers to get up and running.

Download the Distribution

JBoss OSGi is distributed as an IzPack [http://izpack.org] installer archive. The installer is available from the JBoss OSGi download area [http://sourceforge.net/projects/jboss/files/JBossOSGi] .

Running the Installer

To run the installer execute the following command:

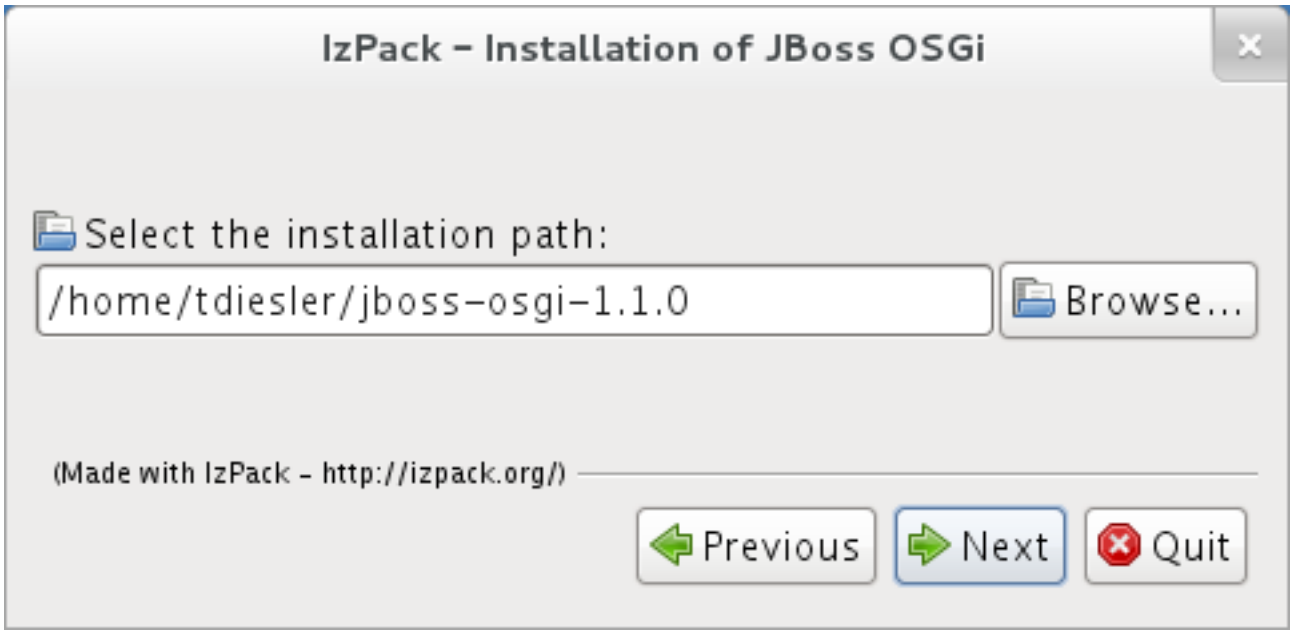
```
java -jar jboss-osgi-installer-1.1.0.jar
```

The installer first shows a welcome screen

Figure 2.1.

Then you select the installation path for the JBoss OSGi distribution. This is the directory where you find the binary build artifacts, the java sources, documentation and the JBoss OSGi Runtime.

Figure 2.2.



The content of the JBoss OSGi distribution contains a set of documents and example test cases that can be executed against the embedded framework or against an AS7 instance.

Activating the Subsystem

By default the OSGi subsystem is activated lazily. It means that the framework will not start up unless you deploy an OSGi bundle. You can activate the OSGi subsystem explicitly by setting the activation property to 'eager'

```
<subsystem xmlns="urn:jboss:domain:osgi:1.2" activation="eager">
```

When you start up the AS7 you should see something like this

```
[tdiesler@tdvaio jboss-as-7.1.0.Final]$ bin/standalone.sh
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /home/tdiesler/git/jboss-as-7.1.0.Final/build/target/jboss-as-7.1.0.

  JAVA: /usr/java/jdk1.6/bin/java

  JAVA_OPTS:  ...

=========================================================================

13:56:32,414 INFO  [org.jboss.modules] JBoss Modules version 1.1.1.GA
13:56:32,700 INFO  [org.jboss.msc] JBoss MSC version 1.0.2.GA
13:56:32,802 INFO  [org.jboss.as] JBAS015899: JBoss AS 7.1.0.Final "Thunder" start
...
13:56:35,357 INFO  JBossOSGi Framework Core - 1.1.5
13:56:35,628 INFO  Install bundle: system.bundle:0.0.0
```

```
...
13:56:36,007 INFO  Install bundle: javax.transaction.api:0.0.0
13:56:36,009 INFO  Install bundle: jboss-osgi-logging:1.0.0
13:56:36,056 INFO  Install bundle: jboss-as-osgi-configadmin:7.1.0.Final
13:56:36,124 INFO  Install bundle: org.apache.felix.log:1.0.0
13:56:36,191 INFO  Install bundle: jbosgi-http-api:1.0.5
13:56:36,191 INFO  Install bundle: org.apache.felix.configadmin:1.2.8
13:56:36,490 INFO  Install bundle: osgi.enterprise:4.2.0.201003190513
13:56:36,683 INFO  Starting bundles for start level: 1
13:56:36,686 INFO  Bundle started: jbosgi-http-api:1.0.5
13:56:36,687 INFO  Bundle started: osgi.enterprise:4.2.0.201003190513
13:56:36,705 INFO  Bundle started: jboss-as-osgi-configadmin:7.1.0.Final
13:56:36,722 INFO  Bundle started: jboss-osgi-logging:1.0.0
13:56:36,758 INFO  Bundle started: org.apache.felix.log:1.0.0
13:56:36,760 INFO  Bundle started: javax.transaction.api:0.0.0
13:56:36,797 INFO  Bundle started: org.apache.felix.configadmin:1.2.8
13:56:36,813 INFO  OSGi Framework started
13:56:36,836 INFO  JBAS015874: JBoss AS 7.1.0.Final "Thunder" started in 4804ms
```

Provided Examples

JBoss OSGi comes with a number of examples that you can build and deploy. Each example deployment is verified by an accompanying test case

- **blueprint** - Basic Blueprint Container examples
- **configadmin** - Configuration Admin example
- **ds** - Declarative Services examples
- **eventadmin** - Event Admin examples
- **http** - HttpService examples
- **interceptor** - Examples that intercept and process bundle metadata
- **jbossas** - Integration examples with non OSGi components (i.e. EJB3, Servlet)
- **jmx** - Standard and extended JMX examples
- **jndi** - Bind objects to the Naming Service
- **jta** - Transaction examples
- **simple** - Simple OSGi examples (start here)
- **webapp** - WebApplication (WAR) examples
- **xml parser** - SAX/DOM parser examples

For more information on these examples, see the Chapter 6, *Provided Examples* section.

Bundle Deployment

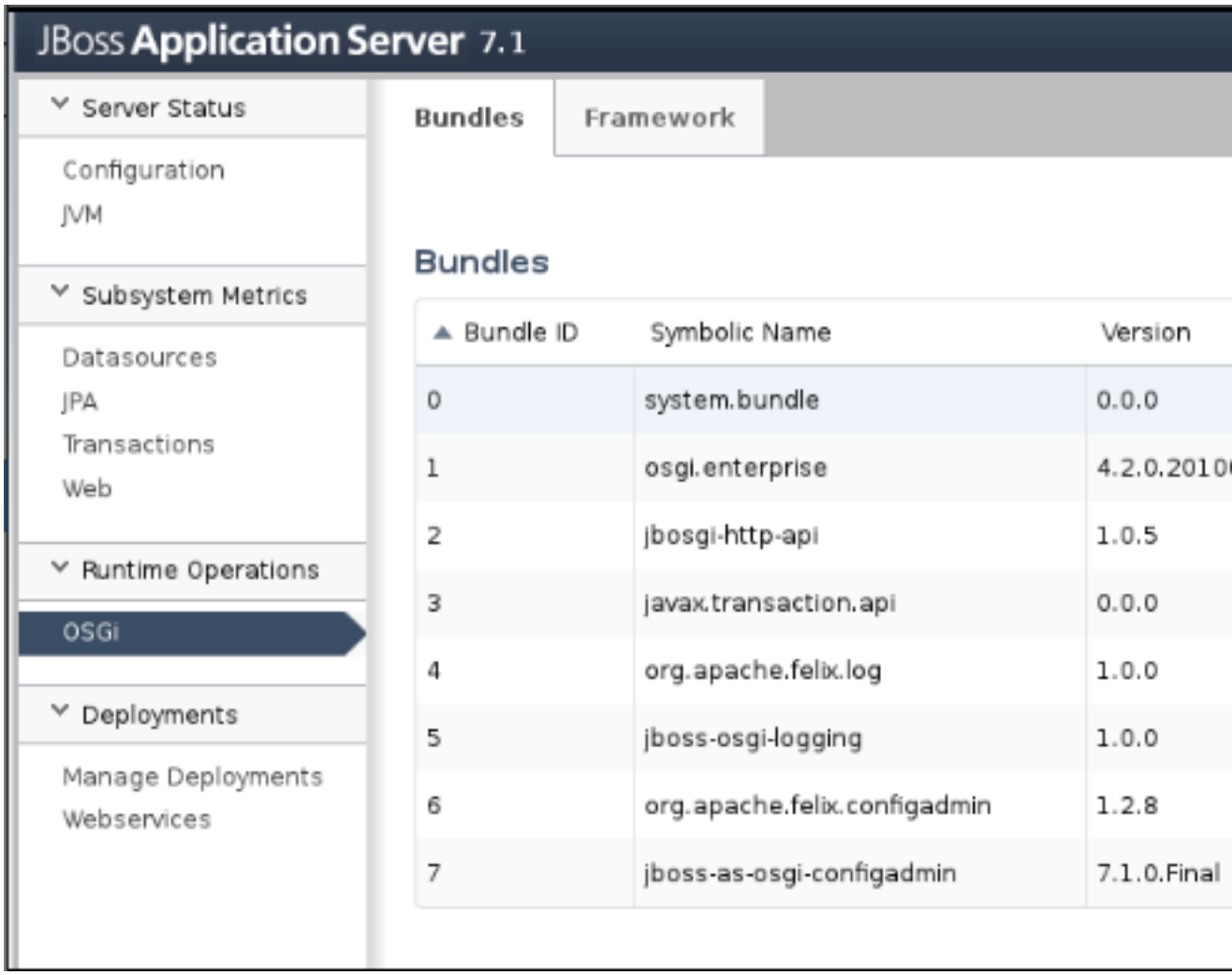
Bundle deployment works, as you would probably expect, by dropping your OSGi Bundle into the **deployments** folder.

```
$ cp org.apache.felix.eventadmin-1.2.6.jar .../jboss-as-7.1.0.Final/standalone/dep
...
14:35:14,319 INFO  JBAS015876: Starting deployment of "org.apache.felix.eventadmin
14:35:14,582 INFO  Install bundle: org.apache.felix.eventadmin:1.2.6
14:35:14,706 INFO  Bundle started: org.apache.felix.eventadmin:1.2.6
14:35:14,777 INFO  JBAS018559: Deployed "org.apache.felix.eventadmin-1.2.6.jar"
```

Managing installed Bundles

JBoss AS7 comes with a Web Console. After startup you can point your browser to http://localhost:9990/console .

Figure 2.3.



The Web Console can be used to install, start, stop and uninstall bundles.

Chapter 3. Application Server Integration

Content

Overview

The JBoss OSGi framework is fully integrated into the JBoss Application Server 7 [http://jboss.org/jbossas] . OSGi bundles can be deployed like any other deployment that is supported by AS. Hot deployment is supported by dropping an OSGi bundle into the 'deployments' folder. JMX and an OSGi management console is also supported.

OSGi components can interact with non OSGi services that are natively provided by AS. This includes, but is not limited to, the Transaction Service and Naming Service (JNDI).

Standard OSGi Config Admin functionality is supported and integrated with the general AS management layer.

By default the OSGi subsystem is activated on-demand. Only when there is an OSGi bundle deployment the subsystem activates and the respective OSGi services become available.

Configuration

The OSGi subsystem is configured like any other subsystem in the standalone/domain XML descriptor. The configuration options are:

- **Subsystem Activation** - By default the OSGi subsystem is activated on-demand. The activation attribute can be set to 'eager' to initialize the subsystem on server startup.
- **Framework Properties** - OSGi supports the notion of framework properties. Property values are of type string. A typical configuration includes a set of packages that are provided by the server directly. Please refer to the OSGi core specification for a list of standard OSGi properties.
- **Module Dependencies** - The Framework can export packages from server system modules. The property 'org.jboss.osgi.system.modules.extra' contains a list of module identifiers that are setup as dependencies of the OSGi Framework.
- **Capabilities** - OSGi bundles can be installed by providing coordinates to the OSGi Repository. Supported coordinates include but are not limited to Maven coordinates and module identifiers.
- **Config Admin properties** - Supported are multiple configurations identified by persistent id (PID). Each configuration may contain multiple configuration key/value pairs.Below is a sample configuration for the OSGi subsystem

```
<subsystem xmlns="urn:jboss:domain:osgi:1.2" activation="lazy">
  <properties>
    <property name="org.jboss.osgi.system.modules.extra">org.apache.log4j</propert
    <property name="org.osgi.framework.system.packages.extra">org.apache.log4j;ver
    <property name="org.osgi.framework.startlevel.beginning">1</property>
  </properties>
  <capabilities>
```

```
    <capability name="javax.servlet.api:v25"/>
    <capability name="javax.transaction.api"/>
    <capability name="org.apache.felix.log" startlevel="1"/>
    <capability name="org.jboss.osgi.logging" startlevel="1"/>
    <capability name="org.apache.felix.configadmin" startlevel="1"/>
    <capability name="org.jboss.as.osgi.configadmin" startlevel="1"/>
  </capabilities>
</subsystem>
...
<subsystem xmlns="urn:jboss:domain:configadmin:1.0">
  <configuration pid="org.apache.felix.webconsole.internal.servlet.OsgiManager">
     <property name="manager.root">jboss-osgi</property>
  </configuration>
</subsystem>
```

For more details on the Application Service integration configuration see AS7 Subsystem Configuration [https://docs.jboss.org/author/pages/viewpage.action?pageId=8094231] documentation.

Features

The current JBoss OSGi feature set in AS includes

- **Blueprint Container Support** - The OSGi Blueprint Container [http://jbossosgi.blogspot.com/2009/04/osgi-blueprint-service-rfc-124.html] allows bundles to contain standard blueprint descriptors, which can be used to create or consume OSGi services. Blueprint components consume OSGi services via injection.
- **ConfigAdmin Support** - ConfigAdmin support is provided by the Apache Felix Configuration Admin Service [http://felix.apache.org/site/apache-felix-config-admin.html] .
- **Declarative Services Support** - Declarative Services support is provided by the Apache Felix Service Component Runtime [http://felix.apache.org/site/apache-felix-service-component-runtime.html] .
- **EventAdmin Support** - EventAdmin support is provided by the Apache Felix Event Admin Service [http://felix.apache.org/site/apache-felix-event-admin.html] .
- **Hot Deployment** - Scans the `deployments` folder for new or removed bundles.
- **HttpService and WebApp Support** - HttpService and WebApp support is provided by Pax Web [http://team.ops4j.org/wiki/display/paxweb/Pax+Web] .
- **JMX Support** - There is local as well as remote JSR160 support for JMX. The OSGi-JMX MBeans are provided through the Apache Aries JMX implementation [http://aries.apache.org] .
- **JNDI Support** - Components can access the JNDI InitialContext as a service from the registry.
- **JTA Support** - Components can interact with the JTA TransactionManager and UserTransaction service.
- **Logging System** - The logging bridge writes OSGi Log Service LogEntries to the server log file.
- **Repository Support** - The OSGi repository can be used to provision the subsystem.
- **XML Parser Support** - The Runtime comes with an implementation of an XMLParserActivator [http://www.osgi.org/javadoc/r4v41/org/osgi/util/xml/XMLParserActivator.html] which provides access to a SAXParserFactory and DocumentBuilderFactory.

Integration Examples

The **JavaEEIntegrationTestCase** deployes two bundles

- example-javaee-api
- example-javaee-service

and two JavaEE archives

- example-javaee-ejb3
- example-javaee-servlet

It demonstrates how JavaEE components can access OSGi services.

```
public void testServletAccess() throws Exception {
   deployer.deploy(API_DEPLOYMENT_NAME);
   deployer.deploy(SERVICE_DEPLOYMENT_NAME);
   deployer.deploy(EJB3_DEPLOYMENT_NAME);
   deployer.deploy(SERVLET_DEPLOYMENT_NAME);

   String response = getHttpResponse("/sample/simple?account=kermit&amount=100", 2
   assertEquals("Calling PaymentService: Charged $100.0 to account 'kermit'", resp

   response = getHttpResponse("/sample/ejb?account=kermit&amount=100", 2000);
   assertEquals("Calling SimpleStatelessSessionBean: Charged $100.0 to account 'ke

   deployer.undeploy(SERVLET_DEPLOYMENT_NAME);
   deployer.undeploy(EJB3_DEPLOYMENT_NAME);
   deployer.undeploy(SERVICE_DEPLOYMENT_NAME);
   deployer.undeploy(API_DEPLOYMENT_NAME);
}
```

The JavaEE components must declare and explicit dependency on OSGi and the API bundle in order to see the service interface.

```
JavaArchive archive = ShrinkWrap.create(JavaArchive.class, EJB3_DEPLOYMENT_NAME);
archive.addClasses(SimpleStatelessSessionBean.class);
archive.setManifest(new Asset() {
  @Override
  public InputStream openStream() {
     ManifestBuilder builder = ManifestBuilder.newInstance();
     String osgidep = "org.osgi.core,org.jboss.osgi.framework";
     String apidep = ",deployment." + API_DEPLOYMENT_NAME + ":0.0.0";
     builder.addManifestHeader("Dependencies", osgidep + apidep);
     return builder.openStream();
  }
});
```

Note, how the API bundles is prefixed with 'deployment' and suffixed with its version in the Dependencies header.

The JavaEE component itself can get the OSGi system BundleContext injected and use it to track the OSGi service it wants to work with.

```
public class SimpleStatelessSessionBean {

   @Resource
   private BundleContext context;

   private PaymentService service;

   @PostConstruct
   public void init() {

        // Track {@link PaymentService} implementations
        ServiceTracker tracker = new ServiceTracker(context, PaymentService.class.

           @Override
           public Object addingService(ServiceReference sref) {
              service = (PaymentService) super.addingService(sref);
              return service;
           }

           @Override
           public void removedService(ServiceReference sref, Object sinst) {
              super.removedService(sref, service);
              service = null;
           }
        };
        tracker.open();
    }

    public String process(String account, String amount) {
        if (service == null) {
            return "PaymentService not available";
        }
        return service.process(account, amount != null ? Float.valueOf(amount) : n
    }
}
```

Chapter 4. Developer Documentation

Content

Bootstrapping JBoss OSGi

OSGiBootstrap provides an OSGiFramework through a OSGiBootstrapProvider.

A OSGiBootstrapProvider is discovered in two stages

1. Read the bootstrap provider class name from a system property

2. Read the bootstrap provider class name from a resource file

In both cases the key is the fully qualified name of the `org.jboss.osgi.spi.framework.OSGiBootstrapProvider` interface.

The following code shows how to get the default `OSGiFramework` from the `OSGiBootstrapProvider` .

```
OSGiBootstrapProvider bootProvider = OSGiBootstrap.getBootstrapProvider();
    OSGiFramework framework = bootProvider.getFramework();
    Bundle bundle = framework.getSystemBundle();
```

The `OSGiBootstrapProvider` can also be configured explicitly. The `OSGiFramework` is a named object from the configuration.

```
OSGiBootstrapProvider bootProvider = OSGiBootstrap.getBootstrapProvider();
    bootProvider.configure(configURL);

    OSGiFramework framework = bootProvider.getFramework();
    Bundle bundle = framework.getSystemBundle();
```

The JBoss OSGi SPI comes with a default bootstrap provider:

- PropertiesBootstrapProvider [http://docs.jboss.org/osgi/apidocs/org/jboss/osgi/spi/framework/PropertiesBootstrapProvider.html]

OSGiBootstrapProvider implementations that read their configurtation from some other source are possible, but currently not part of the JBoss OSGi SPI.

Management View

JBoss OSGi provides standard org.osgi.jmx [http://www.osgi.org/javadoc/r4v42/org/osgi/jmx/package-frame.html] management. Additional to that we provide an MBeanServer [http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanServer.html] service.

Configure AS7 to provide OSGi Management

OSGi Management can be enabled with these capabilities

```
<capabilities>
```

```
  ...
  <capability name="org.apache.aries:org.apache.aries.util:0.4"/>
  <capability name="org.apache.aries.jmx:org.apache.aries.jmx:0.3"/>
  <capability name="org.jboss.osgi.jmx:jbosgi-jmx:1.0.11"/>
</capabilities>
```

Writing Test Cases

Simple Framework Test Case

The most basic form of OSGi testing can be done with an OSGiFrameworkTest. This would boostrap the framework in the @BeforeClass scope and make the framework instance available through getFramework(). Due to classloading restrictions, you can only share primitive types between the test and the framework.

```
public class SimpleFrameworkTestCase extends OSGiFrameworkTest
{
   @Test
   public void testSimpleBundle() throws Exception
   {
      // Get the bundle location
      URL url = getTestArchiveURL("example-simple.jar");

      // Install the Bundle
      BundleContext sysContext = getFramework().getBundleContext();
      Bundle bundle = sysContext.installBundle(url.toExternalForm());
      assertBundleState(Bundle.INSTALLED, bundle.getState());

      // Start the bundle
      bundle.start();
      assertBundleState(Bundle.ACTIVE, bundle.getState());

      // Stop the bundle
      bundle.stop();
      assertBundleState(Bundle.RESOLVED, bundle.getState());

      // Uninstall the bundle
      bundle.uninstall();
      assertBundleState(Bundle.UNINSTALLED, bundle.getState());
   }
}
```

These tests always work with an embedded OSGi framework. You can use the -Dframework property to run the test against a different framework implemenation (i.e. Apache Felix [http://felix.apache.org]).

Lifecycle Interceptors

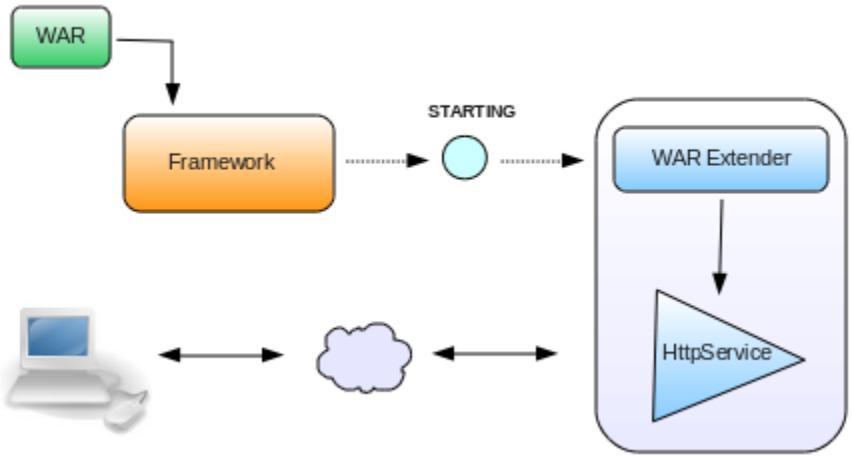
A common pattern in OSGi is that a bundle contains some piece of meta data that gets processed by some other infrastructure bundle that is installed in the OSGi Framework. In such cases the well known Extender Pattern [http://www.osgi.org/blog/2007/02/osgi-extender-model.html] is often being used. JBoss OSGi offeres a differnet approach to address this problem which is covered by the Extender Pattern vs. Lifecycle Interceptor [http://jbossosgi.blogspot.com/2009/10/extender-pattern-vs-lifecycle.html] post in the JBoss OSGi Diary [http://jbossosgi.blogspot.com/] .

Extending an OSGi Bundle

1. Extender registers itself as BundleListener
2. Bundle gets installed/started# Framework fires a BundleEvent
3. Extender picks up the BundleEvent (e.g. STARTING)
4. Extender reads metadata from the Bundle and does its work

There is no extender specific API. It is a pattern rather than a piece of functionality provided by the Framework. Typical examples of extenders are the Blueprint or Web Application Extender.

Figure 4.1.



Client code that installs, starts and uses the registered endpoint could look like this.

```
// Install and start the Web Application bundle
Bundle bundle = context.installBundle("mywebapp.war");
bundle.start();

// Access the Web Application
String response = getHttpResponse("http://localhost:8090/mywebapp/foo");
assertEquals("ok", response);
```

This seemingly trivial code snippet has a number of issues that are probably worth looking into in more detail

- The WAR might have missing or invalid web metadata (i.e. an invalid WEB-INF/web.xml descriptor)
- The WAR Extender might not be present in the system
- There might be multiple WAR Extenders present in the system
- Code assumes that the endpoint is available on return of bundle.start()

Most Blueprint or WebApp bundles are not useful if their Blueprint/Web metadata is not processed. Even if they are processed but in the "wrong" order a user might see unexpected results (i.e. the webapp processes the first request before the underlying Blueprint app is wired together).

As a consequence the extender pattern is useful in some cases but not all. It is mainly useful if a bundle can optionally be extended in the true sense of the word.

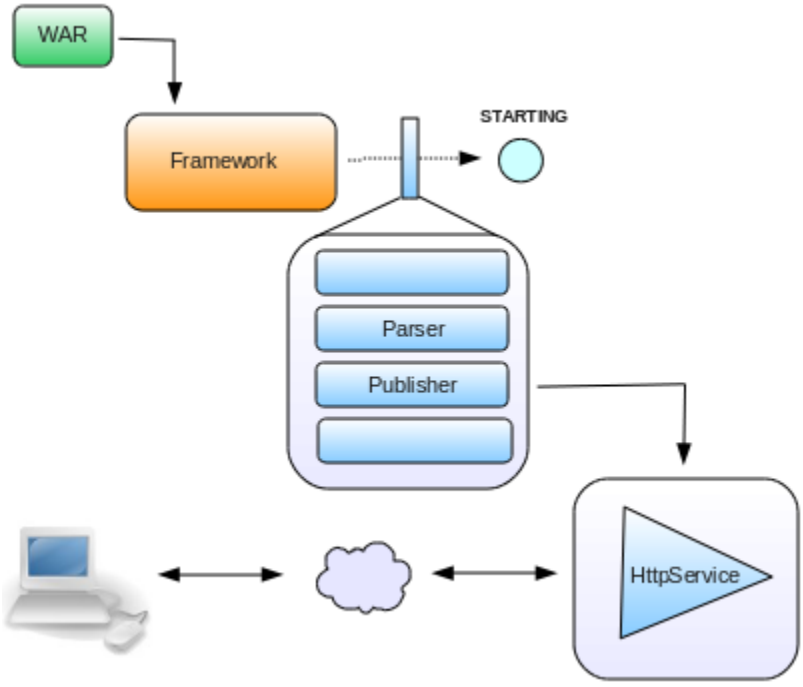
Intercepting the Bundle Lifecycle

If the use case requires the notion of "interceptor" the extender pattern is less useful. The use case might be such that you would want to intercept the bundle lifecycle at various phases to do mandatory metadata processing.

An interceptor could be used for annotation processing, byte code weaving, and other non-optional/ optional metadata processing steps. Typically interceptors have a relative order, can communicate with each other, veto progress, etc.

Lets look at how multiple interceptors can be used to create Web metadata and publish endpoints on the HttpService based on that metadata.

Figure 4.2.



Here is how it works

1. The Wep Application processor registers two LifecycleInterceptors with the LifecycleInterceptorService
2. The Parser interceptor declares no required input and WebApp metadata as produced output
3. The Publisher interceptor declares WebApp metadata as required input
4. The LifecycleInterceptorService reorders all registered interceptors according to their input/output requirements and relative order
5. The WAR Bundle gets installed and started
6. The Framework calls the LifecycleInterceptorService prior to the actual state change

7. The LifecycleInterceptorService calls each interceptor in the chain

8. The Parser interceptor processes WEB-INF/web.xml in the invoke(int state, InvocationContext context) method and attaches WebApp metadata to the InvocationContext

9. The Publisher interceptor is only called when the InvocationContext has WebApp metadata attached. If so, it publishes the endpoint from the WebApp metadata

10. If no interceptor throws an Exception the Framework changes the Bundle state and fires the BundleEvent.

Client code is identical to above.

```
// Install and start the Web Application bundle
Bundle bundle = context.installBundle("mywebapp.war");
bundle.start();

// Access the Web Application
String response = getHttpResponse("http://localhost:8090/mywebapp/foo");
assertEquals("ok", response);
```

The behaviour of that code however, is not only different but also provides a more natural user experience.

- Bundle.start() fails if WEB-INF/web.xml is invalid
- An interceptor could fail if web.xml is not present
- The Publisher interceptor could fail if the HttpService is not present
- Multiple Parser interceptors would work mutually exclusiv on the presents of attached WebApp metadata
- The endpoint is guaranteed to be available when Bundle.start() returns

The general idea is that each interceptor takes care of a particular aspect of processing during state changes. In the example above WebApp metadata might get provided by an interceptor that scans annotations or by another one that generates the metadata in memory. The Publisher interceptor would not know nor care who attached the WebApp metadata object, its task is to consume the WebApp metadata and publish endpoints from it.

For details on howto provide and register liefecycle interceptors have a look at the Lifecycle Interceptor Example.

Chapter 5. Arquillian Test Framework

Content

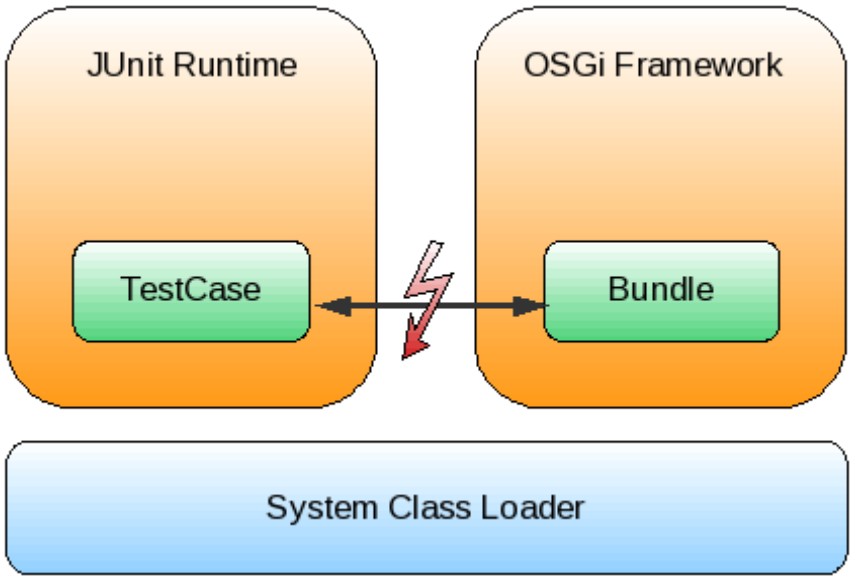
Overview

Arquillian [http://jboss.org/arquillian] is a Test Framework that allows you to run plain JUnit4 test cases from within an OSGi Framework. That the test is actually executed in the the OSGi Framework is transparent to your test case. There is no requirement to extend a specific base class. Your OSGi tests execute along side with all your other (non OSGi specific) test cases in Maven, Ant, or Eclipse.

Some time ago I was looking for ways to test bundles that are deployed to a remote instance of the JBoss OSGi Runtime. I wanted the solution to also work with an OSGi Framework that is bootstrapped from within a JUnit test case.

The basic problem is of course that you cannot access the artefacts that you deploy in a bundle directly from your test case, because they are loaded from different classloaders.

Figure 5.1.



For this release, we extended the Arquillian Test Framework [http://jboss.org/arquillian] to provide support for these requirements.

- Test cases SHOULD be plain JUnit4 POJOs
- There SHOULD be no requirement to extend a specific test base class
- There MUST be no requirement on a specific test runner (i.e. MUST run with Maven)
- There SHOULD be a minimum test framework leakage into the test case
- The test framework MUST support embedded and remote OSGi runtimes with no change required to the test

- The same test case MUST be executable from outside as well as from within the OSGi Framework
- There SHOULD be a pluggable communication layer from the test runner to the OSGi Framework
- The test framework MUST NOT depend on OSGi Framework specific features
- There MUST be no automated creation of test bundles required by the test framework

Configuration

In the target OSGi Framework, you need to have the **arquillian-osgi-bundle.jar** up and running. For remote testing you also need **jboss-osgi-jmx.jar** because Arquillian uses the a standard JSR-160 [http://jcp.org/aboutJava/communityprocess/final/jsr160] to communicate between the test client and the remote OSGi Framework.

See jboss-osgi-jmx on how the JMX protocol can be configured.

Writing Arquillian Tests

In an Arquillian test you

- need to use the **@RunWith(Arquillian.class)** test runner
- may have a **@Deployment** method that generates the test bundle
- may have **@Inject BundleContext** to get the system BundleContext injected
- may have **@Inject Bundle** to get the test bundle injected

```
@RunWith(Arquillian.class)
public class SimpleArquillianTestCase
{
   @Inject
   public Bundle bundle;

   @Deployment
   public static JavaArchive createdeployment()
   {
      final JavaArchive archive = ShrinkWrap.create(JavaArchive.class, "example-ar
      archive.addClasses(SimpleActivator.class, SimpleService.class);
      archive.setManifest(new Asset()
      {
         public InputStream openStream()
         {
            OSGiManifestBuilder builder = OSGiManifestBuilder.newInstance();
            builder.addBundleSymbolicName(archive.getName());
            builder.addBundleManifestVersion(2);
            builder.addBundleActivator(SimpleActivator.class.getName());
            return builder.openStream();
         }
      });
      return archive;
   }
```

```
    @Test
    public void testBundleInjection() throws Exception
    {
       // Assert that the bundle is injected
       assertNotNull("Bundle injected", bundle);

       // Assert that the bundle is in state RESOLVED
       // Note when the test bundle contains the test case it
       // must be resolved already when this test method is called
       assertEquals("Bundle RESOLVED", Bundle.RESOLVED, bundle.getState());

       // Start the bundle
       bundle.start();
       assertEquals("Bundle ACTIVE", Bundle.ACTIVE, bundle.getState());

       // Get the service reference
       BundleContext context = bundle.getBundleContext();
       ServiceReference sref = context.getServiceReference(SimpleService.class.getN
       assertNotNull("ServiceReference not null", sref);

       // Get the service for the reference
       SimpleService service = (SimpleService)context.getService(sref);
       assertNotNull("Service not null", service);

       // Invoke the service
       int sum = service.sum(1, 2, 3);
       assertEquals(6, sum);

       // Stop the bundle
       bundle.stop();
       assertEquals("Bundle RESOLVED", Bundle.RESOLVED, bundle.getState());
    }
}
```

Chapter 6. Provided Examples

Content

Build and Run the Examples

JBoss OSGi comes with a number of examples that demonstrate supported functionality and show best practices. All examples are part of the binary distribution and tightly integrated in our Maven Build Process [http://www.jboss.org/community/docs/DOC-13275] .

The examples can be either run against an embedded OSGi framework or against the AS7 Runtime. Here is how you build and run the against the embedded framework.

```
[tdiesler@tddell example]$ mvn test

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.jboss.test.osgi.example.webapp.WebAppInterceptorTestCase
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 14.417 sec
...

Tests run: 23, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 37.507s
[INFO] Finished at: Wed Mar 07 09:15:50 CET 2012
[INFO] Final Memory: 13M/154M
[INFO] ------------------------------------------------------------------------
```

To run the examples against AS7, you need to provide the target container that the runtime should connect to. This can be done with the **target.container** system property.

```
mvn -Dtarget.container=jboss710 test
```

Blueprint Container

The **BlueprintTestCase** shows how a number of components can be wired together and registered as OSGi service through the Blueprint Container Service.

The example uses this simple blueprint descriptor

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" ...>

  <bean id="beanA" class="org.jboss.test.osgi.example.blueprint.bundle.BeanA">
    <property name="mbeanServer" ref="mbeanService"/>
  </bean>

  <service id="serviceA" ref="beanA" interface="org.jboss.test.osgi.example.bluepr
```

```
  </service>

  <service id="serviceB" interface="org.jboss.test.osgi.example.blueprint.bundle.S
    <bean class="org.jboss.test.osgi.example.blueprint.bundle.BeanB">
       <property name="beanA" ref="beanA"/>
    </bean>
  </service>

  <reference id="mbeanService" interface="javax.management.MBeanServer"/>

</blueprint>
```

The Blueprint Container registers two services **ServiceA** and **ServiceB** . ServiceA is backed up by **BeanA** , ServiceB is backed up by the anonymous **BeanB** . BeanA is injected into BeanB and the **MBeanServer** gets injected into BeanA. Both beans are plain POJOs. There is **no BundleActivator** neccessary to register the services.

The example test verifies the correct wiring like this

```
@Test
public void testServiceA() throws Exception
{
  ServiceReference sref = context.getServiceReference(ServiceA.class.getName());
  assertNotNull("ServiceA not null", sref);

  ServiceA service = (ServiceA)context.getService(sref);
  MBeanServer mbeanServer = service.getMbeanServer();
  assertNotNull("MBeanServer not null", mbeanServer);
}

@Test
public void testServiceB() throws Exception
{
  ServiceReference sref = context.getServiceReference(ServiceB.class.getName());
  assertNotNull("ServiceB not null", sref);

  ServiceB service = (ServiceB)context.getService(sref);
  BeanA beanA = service.getBeanA();
  assertNotNull("BeanA not null", beanA);
}
```

Blueprint support in AS7

This test uses the OSGi Repository functionality to provision the runtime with the required support functionality like this

```
ManagementSupport.provideMBeanServer(context, bundle);
   BlueprintSupport.provideBlueprint(context, bundle);
```

To enable blueprint support in AS7 you would configure these capabilities

```
<capability name="org.apache.aries:org.apache.aries.util:0.4"/>
<capability name="org.apache.aries.proxy:org.apache.aries.proxy:0.4"/>
<capability name="org.apache.aries.blueprint:org.apache.aries.blueprint:0.4"/>
```

Configuration Admin

The **ConfigurationAdminTestCase** shows how an OSGi ManagedService [http://www.osgi.org/javadoc/r4v42/org/osgi/service/cm/ManagedService.html] can be configured through the ConfigurationAdmin [http://www.osgi.org/javadoc/r4v42/org/osgi/service/cm/ConfigurationAdmin.html] service.

```
public void testManagedService() throws Exception {

   // Get the {@link Configuration} for the given PID
   Configuration config = configAdmin.getConfiguration(ConfiguredService.SERVICE_P
   assertNotNull("Config not null", config);

   Dictionary<String, String> configProps = new Hashtable<String, String>();
   configProps.put("foo", "bar");
   config.update(configProps);

   // Register a {@link ManagedService}
   Dictionary<String, String> serviceProps = new Hashtable<String, String>();
   serviceProps.put(Constants.SERVICE_PID, ConfiguredService.SERVICE_PID);
   bundlecontext.registerService(new String[] { ConfiguredService.class.getName(),
      ManagedService.class.getName() }, new ConfiguredService(), serviceProps);

   // Wait a little for the update event
   if (latch.await(5, TimeUnit.SECONDS) == false)
      throw new TimeoutException();

   // Verify service property
   ServiceReference sref = bundlecontext.getServiceReference(ConfiguredService.cla
   ConfiguredService service = (ConfiguredService) bundlecontext.getService(sref);
   assertEquals("bar", service.getValue("foo"));

   config.delete();
}
```

Configuration Admin support in AS7

Configuration Admin support is build into the **config admin** subsystem and is available by default. The OSGi configurations will appear together with any other configurations that use this service in the AS7 domain model.

For the OSGi specific part we use this capability, which is also configured by default

```
<capability name="org.apache.felix.configadmin"/>
```

Declarative Services

The **DeclarativeServicesTestCase** shows how a service can be made available through a Declarative Services descriptor.

```
<component name="sample.component" immediate="true">
  <implementation class="org.jboss.test.osgi.example.ds.SampleComparator" />
  <property name="service.description" value="Sample Comparator Service" />
  <property name="service.vendor" value="Apache Software Foundation" />
```

```
  <service>
    <provide interface="java.util.Comparator" />
  </service>
</component>
```

The test then verifies that the service becomes available

```
public void testImmediateService() throws Exception {

  // Track the service provided by the test bundle
  final CountDownLatch latch = new CountDownLatch(1);
  ServiceTracker tracker = new ServiceTracker(context, Comparator.class.getName(),
     public Object addingService(ServiceReference reference) {
         Comparator<Object> service = (Comparator<Object>) super.addingService(ref
         latch.countDown();
         return service;
     }
  };
  tracker.open();

  // Wait for the service to become available
  if (latch.await(2, TimeUnit.SECONDS) == false)
    throw new TimeoutException("Timeout tracking Comparator service");
}
```

Declarative Services support in AS7

This test uses the OSGi Repository functionality to provision the runtime with the required support functionality like this

```
DeclarativeServicesSupport.provideDeclarativeServices(context, bundle);
```

To enable declarative services support in AS7 you would configure this capability

```
<capability name="org.apache.felix:org.apache.felix.scr:1.6.0"/>
```

Event Admin

The **EventAdminTestCase** uses the EventAdmin [http://www.osgi.org/javadoc/r4v42/org/osgi/service/event/EventAdmin.html] service to send/receive events.

```
public void testEventHandler() throws Exception
{
  TestEventHandler eventHandler = new TestEventHandler();

  // Register the EventHandler
  Dictionary param = new Hashtable();
  param.put(EventConstants.EVENT_TOPIC, new String[Introduction] { TOPIC });
  context.registerService(EventHandler.class.getName(), eventHandler, param);

  // Send event through the the EventAdmin
  EventAdmin eventAdmin = EventAdminSupport.provideEventAdmin(context, bundle);
  eventAdmin.sendEvent(new Event(TOPIC, null));
```

```
  // Verify received event
  assertEquals("Event received", 1, eventHandler.received.size());
  assertEquals(TOPIC, eventHandler.received.get(0).getTopic());
}
```

Event Admin support in AS7

This test uses the OSGi Repository functionality to provision the runtime with the required support functionality like this

```
EventAdminSupport.provideEventAdmin(context, bundle);
```

To enable event admin support in AS7 you would configure this capability

```
<capability name="org.apache.felix:org.apache.felix.eventadmin:1.2.6"/>
```

HttpService

The **HttpServiceTestCase** deploys a Service that registeres a servlet and a resource with the HttpService [http://www.osgi.org/javadoc/r4v41/org/osgi/service/http/HttpService.html] .

```
ServiceTracker tracker = new ServiceTracker(context, HttpService.class.getName(),
tracker.open();

HttpService httpService = (HttpService)tracker.getService();
if (httpService == null)
   throw new IllegalStateException("HttpService not registered");

Properties initParams = new Properties();
initParams.setProperty("initProp", "SomeValue");
httpService.registerServlet("/servlet", new EndpointServlet(context), initParams,
httpService.registerResources("/file", "/res", null);
```

The test then verifies that the registered servlet context and the registered resource can be accessed.

HttpService support in AS7

This test uses the OSGi Repository functionality to provision the runtime with the required support functionality like this

```
HttpServiceSupport.provideHttpService(context, bundle);
```

To enable HttpService support in AS7 you would configure this capability

```
<capability name="org.ops4j.pax.web:pax-web-jetty-bundle:1.1.2"/>
```

JMX Service

MBeanServer Service

The **MBeanServerTestCase** tracks the MBeanServer service and registers a pojo with JMX. It then verifies the JMX access.

```
public class MBeanActivator implements BundleActivator
{
   public void start(BundleContext context)
   {
      ServiceTracker tracker = new ServiceTracker(context, MBeanServer.class.getNa
      {
         public Object addingService(ServiceReference reference)
         {
            MBeanServer mbeanServer = (MBeanServer)super.addingService(reference);
            registerMBean(mbeanServer);
            return mbeanServer;
         }

         @Override
         public void removedService(ServiceReference reference, Object service)
         {
            unregisterMBean((MBeanServer)service);
            super.removedService(reference, service);
         }
      };
      tracker.open();
   }

   public void stop(BundleContext context)
   {
      ServiceReference sref = context.getServiceReference(MBeanServer.class.getNam
      if (sref != null)
      {
         MBeanServer mbeanServer = (MBeanServer)context.getService(sref);
         unregisterMBean(mbeanServer);
      }
   }
   ...
}

public void testMBeanAccess() throws Exception
{
   // Provide MBeanServer support
   MBeanServer server = ManagementSupport.provideMBeanServer(context, bundle);

   // Start the test bundle
   bundle.start();

   ObjectName oname = ObjectName.getInstance(FooMBean.MBEAN_NAME);
   FooMBean foo = ManagementSupport.getMBeanProxy(server, oname, FooMBean.class);
   assertEquals("hello", foo.echo("hello"));
}
```

Bundle State control via BundleStateMBean

The **BundleStateTestCase** uses JMX to control the bundle state through the BundleStateMBean.

```
public void testBundleStateMBean() throws Exception
{
```

```
   MBeanServer server = ManagementSupport.provideMBeanServer(context, bundle);

   ObjectName oname = ObjectName.getInstance(BundleStateMBean.OBJECTNAME);
   BundleStateMBean bundleState =  ManagementSupport.getMBeanProxy(server, oname,
   assertNotNull("BundleStateMBean not null", bundleState);

   TabularData bundleData = bundleState.listBundles();
   assertNotNull("TabularData not null", bundleData);
   assertFalse("TabularData not empty", bundleData.isEmpty());
}
```

OSGi JMX support in AS7

This test uses the OSGi Repository functionality to provision the runtime with the required support functionality like this

```
ManagementSupport.provideMBeanServer(context, bundle);
```

To enable OSGi JMX support in AS7 you would configure these capabilities

```
<capability name="org.apache.aries:org.apache.aries.util:0.4"/>
<capability name="org.apache.aries.jmx:org.apache.aries.jmx:0.3"/>
```

The MBeanServer service is provided by default in AS7.

JTA Service

The **TransactionTestCase** gets the UserTransaction [http://java.sun.com/javaee/5/docs/api/javax/transaction/UserTransaction.html] service and registers a transactional user object (i.e. one that implements Synchronization [http://java.sun.com/javaee/5/docs/api/javax/transaction/Synchronization.html]) with the TransactionManager [http://java.sun.com/javaee/5/docs/api/javax/transaction/TransactionManager.html] service. It then verifies that modifications on the user object are transactional.

This functionality is only available in the context of AS7.

```
Transactional txObj = new Transactional();

ServiceReference userTxRef = context.getServiceReference(UserTransaction.class.get
assertNotNull("UserTransaction service not null", userTxRef);

UserTransaction userTx = (UserTransaction)context.getService(userTxRef);
assertNotNull("UserTransaction not null", userTx);

userTx.begin();
try
{
   ServiceReference tmRef = context.getServiceReference(TransactionManager.class.g
   assertNotNull("TransactionManager service not null", tmRef);

   TransactionManager tm = (TransactionManager)context.getService(tmRef);
   assertNotNull("TransactionManager not null", tm);

   Transaction tx = tm.getTransaction();
```

```
   assertNotNull("Transaction not null", tx);

   tx.registerSynchronization(txObj);

   txObj.setMessage("Donate $1.000.000");
   assertNull("Uncommited message null", txObj.getMessage());

   userTx.commit();
}
catch (Exception e)
{
   userTx.setRollbackOnly();
}

assertEquals("Donate $1.000.000", txObj.getMessage());

class Transactional implements Synchronization
{
  public void afterCompletion(int status)
  {
     if (status == Status.STATUS_COMMITTED)
        message = volatileMessage;
  }

  ...
}
```

Transaction support in AS7

The related services are provided by default in AS7. The transaction APIs are provided by this capability.

```
<capability name="javax.transaction.api"/>
```

Lifecycle Interceptor

The **LifecycleInterceptorTestCase** deployes a bundle that contains some metadata and an interceptor bundle that processes the metadata and registeres an http endpoint from it. The idea is that the bundle does not process its own metadata. Instead this work is delegated to some specialized metadata processor (i.e. the interceptor).

Each interceptor is itself registered as a service. This is the well known Whiteboard Pattern [http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf] .

```
public class InterceptorActivator implements BundleActivator
{
   public void start(BundleContext context)
   {
      LifecycleInterceptor publisher = new PublisherInterceptor();
      LifecycleInterceptor parser = new ParserInterceptor();

      // Add the interceptors, the order of which is handles by the service
      context.registerService(LifecycleInterceptor.class.getName(), publisher, nul
      context.registerService(LifecycleInterceptor.class.getName(), parser, null);
   }
```

```
}

public class ParserInterceptor extends AbstractLifecycleInterceptor
{
   ParserInterceptor()
   {
      // Add the provided output
      addOutput(HttpMetadata.class);
   }

   public void invoke(int state, InvocationContext context)
   {
      // Do nothing if the metadata is already available
      HttpMetadata metadata = context.getAttachment(HttpMetadata.class);
      if (metadata != null)
         return;

      // Parse and create metadta on STARTING
      if (state == Bundle.STARTING)
      {
          VirtualFile root = context.getRoot();
          VirtualFile propsFile = root.getChild("/http-metadata.properties");
          if (propsFile != null)
          {
             log.info("Create and attach HttpMetadata");
             metadata = createHttpMetadata(propsFile);
             context.addAttachment(HttpMetadata.class, metadata);
          }
      }
   }
   ...
}

public class PublisherInterceptor extends AbstractLifecycleInterceptor
{
   PublisherInterceptor()
   {
      // Add the required input
      addInput(HttpMetadata.class);
   }

   public void invoke(int state, InvocationContext context)
   {
      // HttpMetadata is guaratied to be available because we registered
      // this type as required input
      HttpMetadata metadata = context.getAttachment(HttpMetadata.class);

      // Register HttpMetadata on STARTING
      if (state == Bundle.STARTING)
      {
         String servletName = metadata.getServletName();

         // Load the endpoint servlet from the bundle
         Bundle bundle = context.getBundle();
```

```
            Class servletClass = bundle.loadClass(servletName);
            HttpServlet servlet = (HttpServlet)servletClass.newInstance();

            // Register the servlet with the HttpService
            HttpService httpService = getHttpService(context, true);
            httpService.registerServlet("/servlet", servlet, null, null);
         }

         // Unregister the endpoint on STOPPING
         else if (state == Bundle.STOPPING)
         {
            log.info("Unpublish HttpMetadata: " + metadata);
            HttpService httpService = getHttpService(context, false);
            if (httpService != null)
               httpService.unregister("/servlet");
         }
      }
   }
```

Interceptor support in AS7

This functionality is build into the JBoss OSGi Framework. There is no additional configuration needed in AS7.

Web Application

The **WebAppTestCase** deploys an OSGi Web Application Bundle (WAB). Similar to HTTP Service Example it registers a servlet and resources with the WebApp container. This is done through a standard web.xml descriptor.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" ... version="2.5">

  <display-name>WebApp Sample</display-name>

  <servlet>
    <servlet-name>servlet</servlet-name>
    <servlet-class>org.jboss.test.osgi.example.webapp.bundle.EndpointServlet</serv
    <init-param>
      <param-name>initProp</param-name>
      <param-value>SomeValue</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>servlet</servlet-name>
    <url-pattern>/servlet</url-pattern>
  </servlet-mapping>

</web-app>
```

The associated OSGi manifest looks like this.

```
Manifest-Version: 1.0
```

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: example-webapp
Bundle-ClassPath: .,WEB-INF/classes
Web-ContextPath: example-webapp
Import-Package: javax.servlet,javax.servlet.http,...
```

The test verifies that we can access the servlet and some resources.

```
public void testServletAccess() throws Exception
{
   // Provide WebApp support
   WebAppSupport.provideWebappSupport(context, bundle);

   // Start the test bundle
   bundle.start();

   String line = getHttpResponse("/example-webapp/servlet?test=plain", 5000);
   assertEquals("Hello from Servlet", line);
}
```

OSGi Web Application support in AS7

This test uses the OSGi Repository functionality to provision the runtime with the required support functionality like this

```
WebAppSupport.provideWebappSupport(context, bundle);
```

To enable OSGi Web Application support in AS7 you would configure these capabilities

```
<capability name="org.ops4j.pax.web:pax-web-jetty-bundle:1.1.2"/>
<capability name="org.ops4j.pax.web:pax-web-jsp:1.1.2"/>
<capability name="org.ops4j.pax.web:pax-web-extender-war:1.1.2"/>
```

XML Parser Services

The XML parser test cases get a DocumentBuilderFactory/SAXParserFactory respectivly and unmarshalls an XML document using that parser.

```
DocumentBuilderFactory factory = XMLParserSupport.provideDocumentBuilderFactory(co
factory.setValidating(false);

DocumentBuilder domBuilder = factory.newDocumentBuilder();
URL resURL = context.getBundle().getResource("example-xml-parser.xml");
Document dom = domBuilder.parse(resURL.openStream());
assertNotNull("Document not null", dom);

SAXParserFactory factory = XMLParserSupport.provideSAXParserFactory(context, bundl
factory.setValidating(false);

SAXParser saxParser = factory.newSAXParser();
URL resURL = context.getBundle().getResource("example-xml-parser.xml");

SAXHandler saxHandler = new SAXHandler();
saxParser.parse(resURL.openStream(), saxHandler);
```

```
assertEquals("content", saxHandler.getContent());
```

XML parser support in AS7

This test uses the OSGi Repository functionality to provision the runtime with the required support functionality like this

```
XMLParserSupport.provideDocumentBuilderFactory(context, bundle);
  XMLParserSupport.provideSAXParserFactory(context, bundle);
```

To enable OSGi Web Application support in AS7 you would configure this capability

```
<capability name="org.jboss.osgi.xerces:jbosgi-xerces:2.10.0"/>
```

Chapter 7. References

Resources

- OSGi 4.2 Specifications [http://www.osgi.org/Download/Release4V42]
- OSGi 4.2 JavaDoc [http://www.osgi.org/javadoc/r4v42/]
- OSGi on Wikipedia [http://en.wikipedia.org/wiki/OSGi_Specification_Implementations]
- JBoss OSGi Project Page [http://www.jboss.org/jbossas/osgi]
- JBoss OSGi Wiki [https://docs.jboss.org/author/display/JBOSGI]
- JBoss OSGi Diary [http://jbossosgi.blogspot.com]
- Issue Tracking [https://jira.jboss.org/jira/browse/JBOSGI]
- Source Repository [http://github.com/jbosgi]
- User Forum [http://www.jboss.org/index.html?module=bb&op=viewforum&f=257]
- Design Forum [http://www.jboss.org/index.html?module=bb&op=viewforum&f=256]

Authors

- `<Thomas Diesler>`
- `<David Bosschaert>`

Chapter 8. Getting Support

We offer free support through the JBoss OSGi User Forum [http://www.jboss.org/index.html?module=bb&op=viewforum&f=257] .

Please note, that posts to this forum will be dealt with at the community's leisure. If your business is such that you need to rely on qualified answers within a known time frame, this forum might not be your preferred support channel.

For professional support please go to JBoss Support Services [http://www.jboss.com/services] .

Appendix A. Revision History

Revision History
Revision ToDo 0-0 ToDo Wed Jan 19 2011 ToDo DudeToDo McPants<ToDo Dude.McPants@example.com

ToDo Initial creation of book