JBoss OSGi Home

Table of Contents

1 News

- 08-Mar-2012 JBossOSGi-1.1.0 Released
- 26-Jul-2011 JBossOSGi-1.0.0 Released

2 Goals

JBoss OSGi provides the OSGi 4.2 framework in the JBoss Application Server. Like the AS foundation itself, it is based on the JBoss Modules modularity layer. With integration taking place at the foundation layer, we provide integration of the traditional EE component models with OSGi services.

The OSGi specifications define a standardized, component-oriented, computing environment for networked services that is the foundation of an enhanced service-oriented architecture.

The OSGi specification defines:

- A set of services and APIs that an OSGi container must implement
- A contract between the container and your application
- The Bundle deployment format

OSGi bundles can be deployed in AS 7 alongside other deployments supported by the Application Server.

3 Quick Links

- Project Home
- Downloads
- OSGi API
- OSGi Specifications
- User Forum
- Developer Forum
- OSGi Diary

4 References

- OSGi Service Platform Release 4 Version 4.2
- OSGi Business Whitepaper
- OSGi Technical Whitepaper
- OSGi Best Practices

5 User Guide

5.1 Introduction

5.1.1 Content

5.1.2 What is OSGi

The OSGi specifications 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.

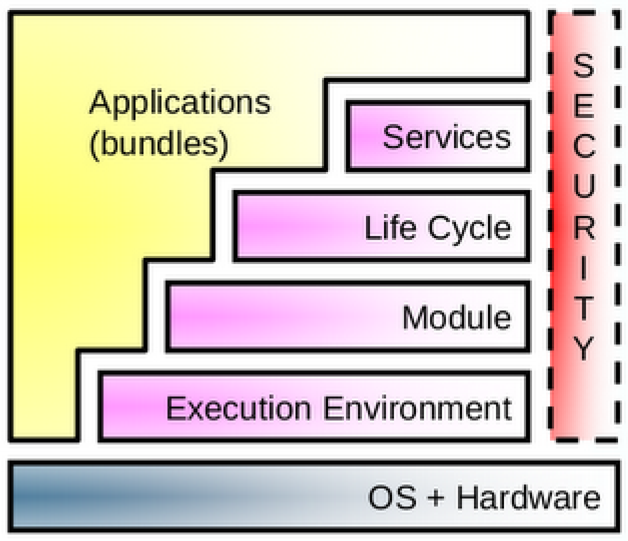
What kind of applications benefit from OSGi?

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.

5.1.3 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



Source: OSGi Alliance

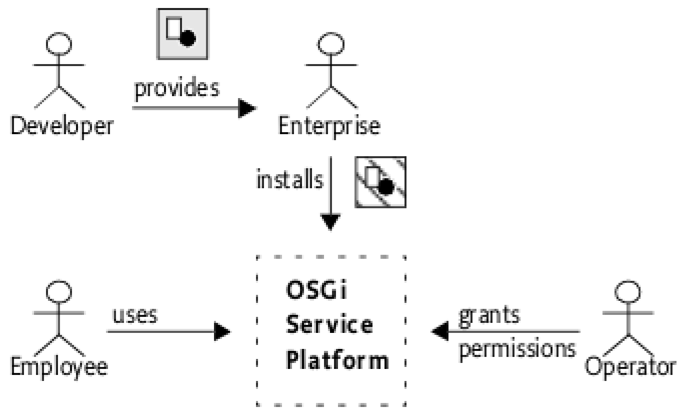
OSGi Security Layer

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.



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.

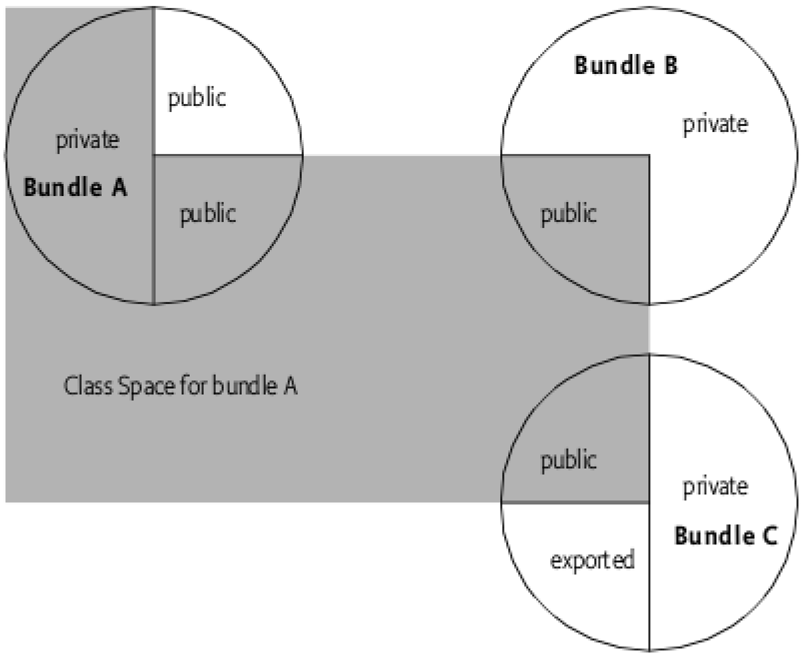
Once a Bundle is started, its functionality is provided and services are exposed to other bundles installed in the OSGi Service Platform. A bundle carries descriptive information about itself in the manifest file that is contained in its JAR file. Here are a few important Manifest Headers defined by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

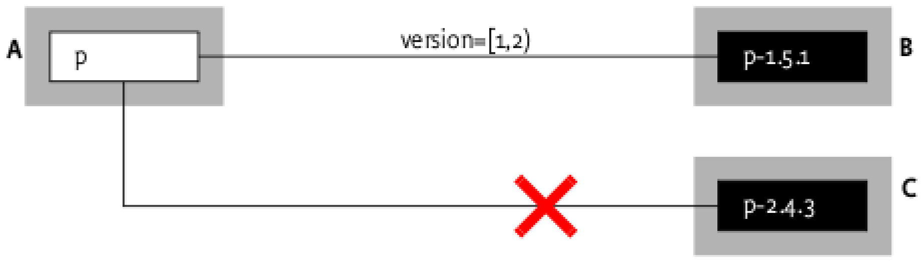
With the OSGi Class Loading Architecture many bundles can share a single virtual machine (VM). Within this VM, bundles can hide packages and classes from other bundles, as well as share packages with other bundles.



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
```



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
```

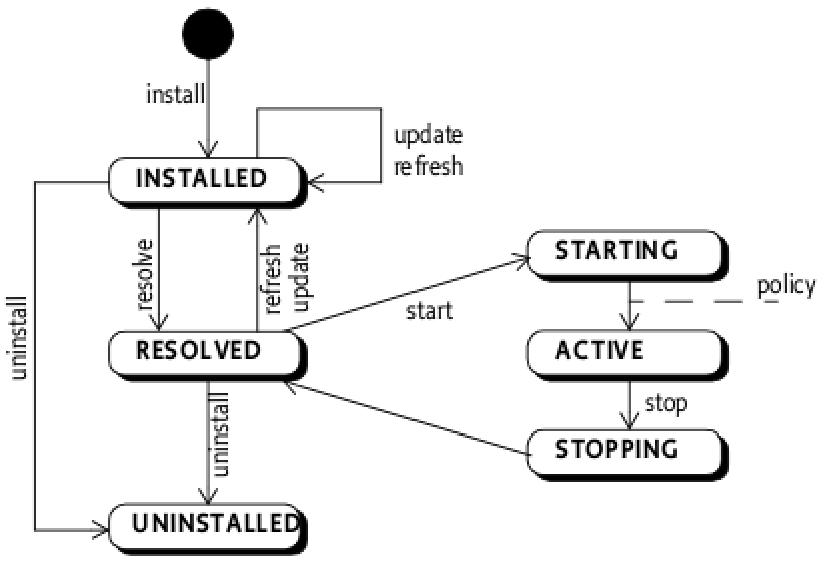
An exporter can limit the visibility of the classes in a package with the include and exclude directives on the export definition.

```
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:



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.

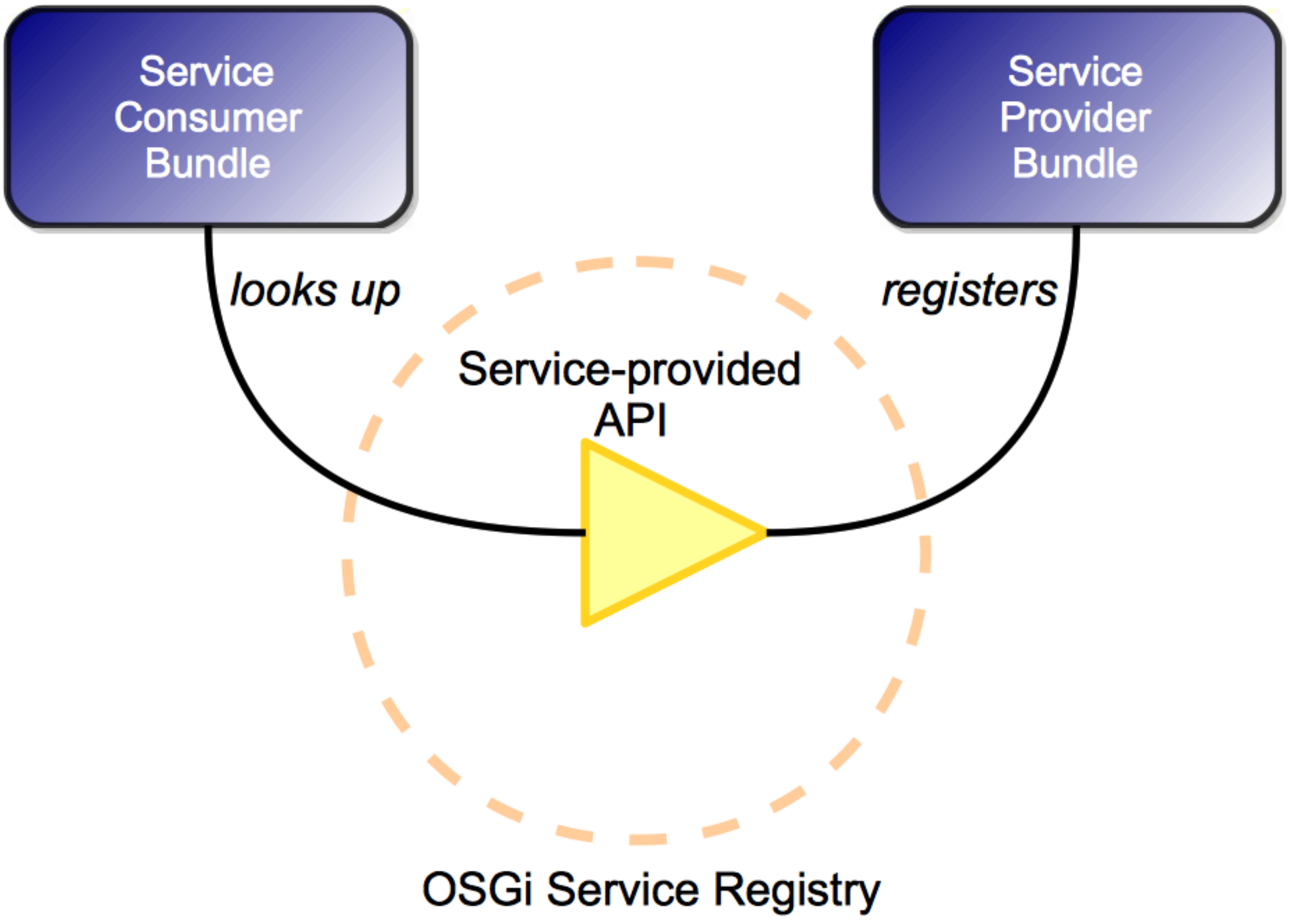
A Bundle Context object represents the execution context of a single bundle within the OSGi Service Platform, and acts as a proxy to the underlying Framework. A *Bundle Context* object is created by the Framework when a bundle is started. The bundle can use this private BundleContext object for the following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



OSGi Services

5.1.4 OSGi Service Compendium

The OSGi Service Compendium is described in the OSGi Compendium and Enterprise specifications. 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

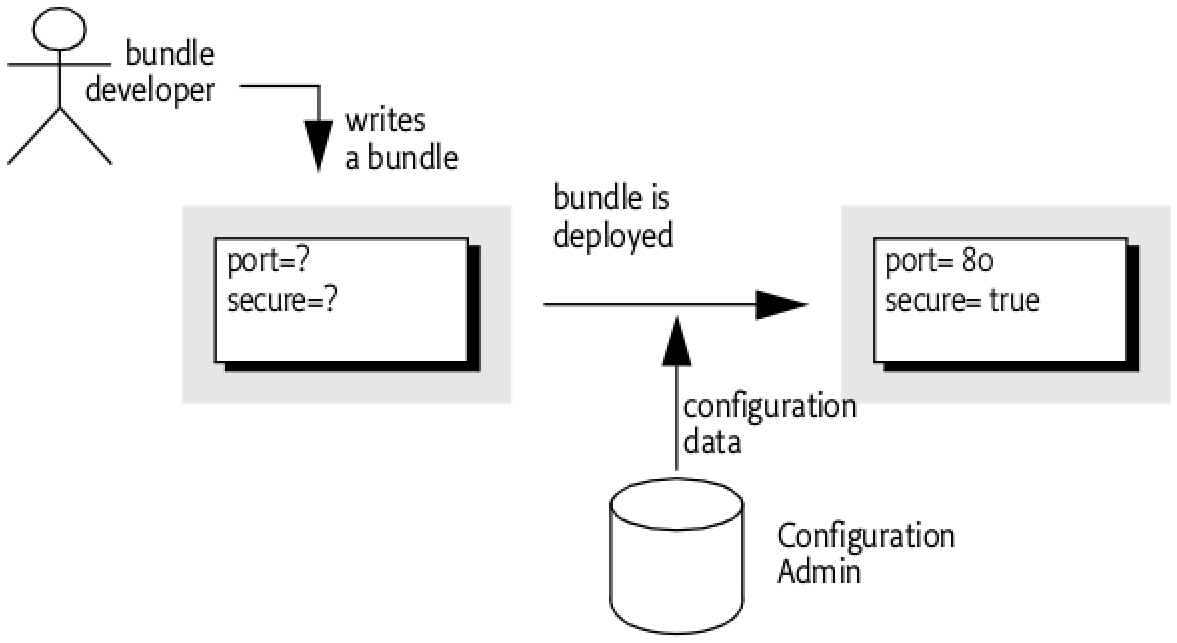
Chapter 102 in the Compendium and Enterprise specifications.

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.



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.

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 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.

5.2 Getting Started

5.2.1 Content

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

5.2.2 Download the Distribution

JBoss OSGi is shipped as part of JBoss Application Server 7, which can be downloaded from the AS 7 download area.

The JBoss OSGi Project Distribution

While AS 7 has OSGi support built-in, the JBoss OSGi project also provides an installer which contains examples and documentation and can in some cases provide more recent releases of the OSGi project than can be found in AS 7.

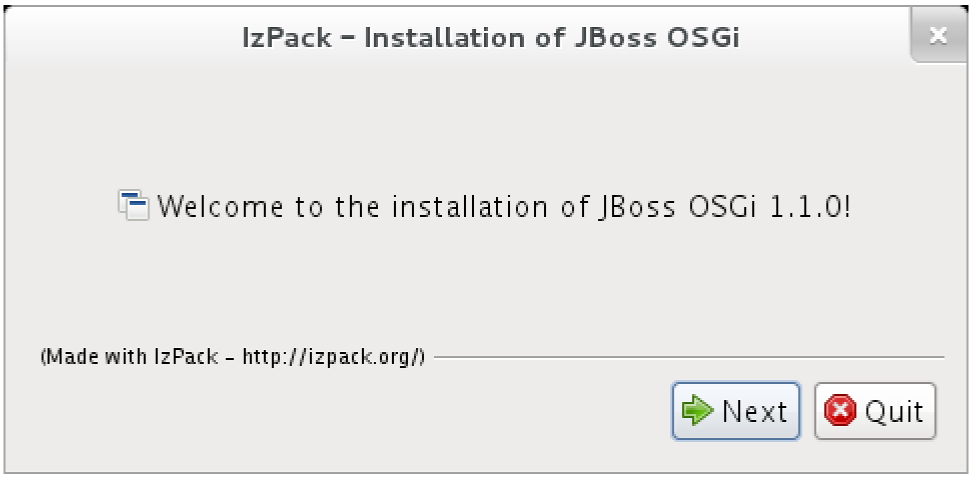
JBoss OSGi is distributed as an IzPack installer archive. The installer is available from the JBoss OSGi download area.

Running the Installer

To run the Project Distribution installer execute the following command:

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

The installer first shows a welcome screen

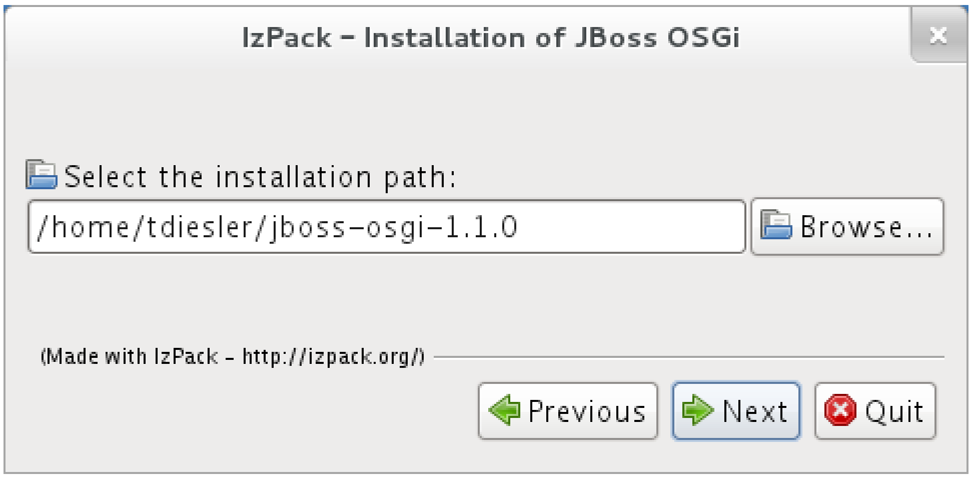


IzPack - Installation of JBoss OSGi

Welcome to the installation of JBoss OSGi 1.1.0!

(Made with IzPack - http://izpack.org/)

Then you select the installation path for the JBoss OSGi distribution. This is the directory where you find binaries, sources, examples and documentation.



IzPack - Installation of JBoss OSGi

Select the installation path:

/home/tdiesler/jboss-osgi-1.1.0

(Made with IzPack - http://izpack.org/)

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.

5.2.3 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.Final

  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" starting
...
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
```

5.2.4 Provided Examples

The JBoss OSGi project distribution 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 Provided Examples section.

5.2.5 Bundle Deployment

Bundle Deployment from the command-line is supported through the CLI console. Bundles can also be deployed using the web-based Management Console.

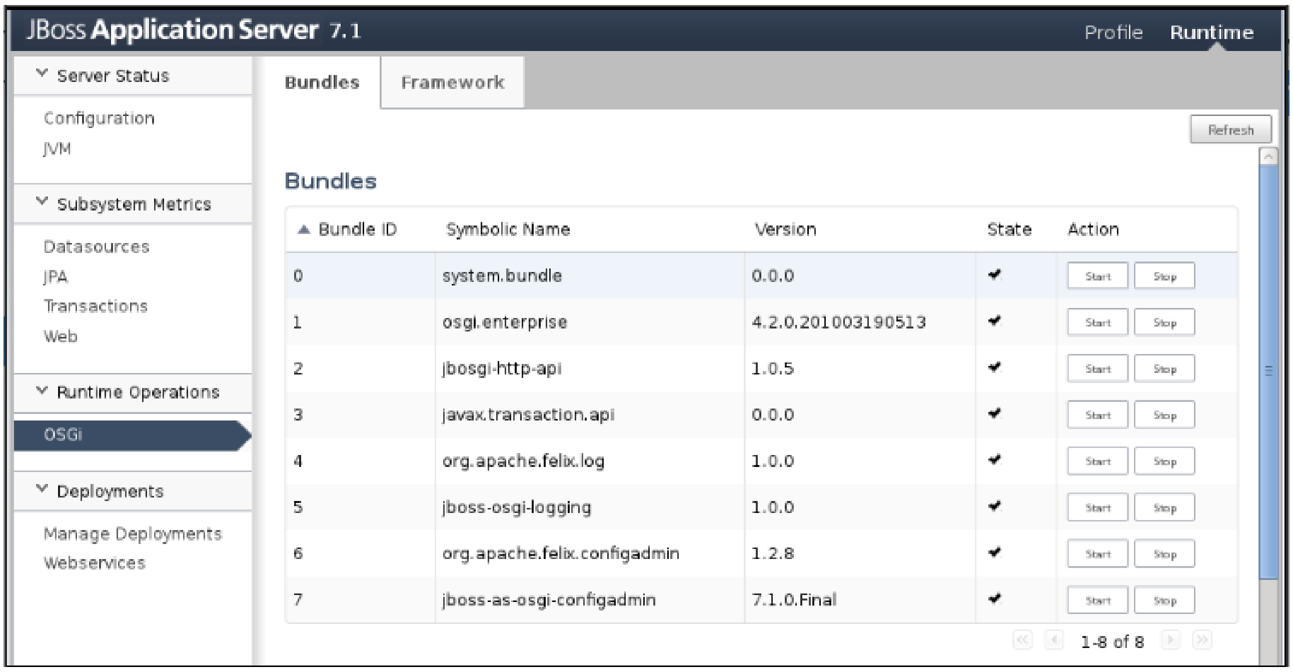
It is also possible to deploy bundles by dropping them in the `deployments` folder, however this will place the deployed bundles in the INSTALLED state. Bundles still need to be started before they become active. Bundles can be started through the CLI or the web console.

When a bundle is installed and started, messages like the following can be seen from the server console.

```
$ bin/standalone.sh
...
14:35:14,319 INFO  JBAS015876: Starting deployment of "org.apache.felix.eventadmin-1.2.6.jar"
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"
```

5.2.6 Managing installed Bundles

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



Bundles

▲ Bundle ID	Symbolic Name	Version	State	Action
0	system.bundle	0.0.0	✓	Start Stop
1	osgi.enterprise	4.2.0.201003190513	✓	Start Stop
2	jbosgi-http-api	1.0.5	✓	Start Stop
3	javax.transaction.api	0.0.0	✓	Start Stop
4	org.apache.felix.log	1.0.0	✓	Start Stop
5	jboss-osgi-logging	1.0.0	✓	Start Stop
6	org.apache.felix.configadmin	1.2.8	✓	Start Stop
7	jboss-as-osgi-configadmin	7.1.0.Final	✓	Start Stop

The Web Console can be used to install, start, stop and uninstall bundles. Additionally, the web console can be used to activate the framework and manage the current framework Start Level.

Felix Web Console

For more detailed management operations you can also install the Felix Web Console

In **7.1.1.Final**, **7.1.2.Final** add these capabilities

```
<capability name="org.ops4j.pax.web:pax-web-jetty-bundle:1.1.2" startlevel="1"/>
<capability name="org.ops4j.pax.web:pax-web-jsp:1.1.2" startlevel="1"/>
<capability name="org.apache.felix:org.apache.felix.webconsole:3.1.8" startlevel="1"/>
```

The console is then accessible on: http://localhost:8090/system/console

In **7.2.0.Alpha1** add this capability

```
<capability name="org.apache.felix:org.apache.felix.webconsole:3.1.8" startlevel="1"/>
```

The console is then accessible on: http://localhost:8080/httpservice/system/console

5.3 Application Server Integration

5.3.1 Content

5.3.2 Overview

The JBoss OSGi framework is fully integrated into the JBoss Application Server 7. 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. Management is supported through the Command Line Console or the web-based Management Console.

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.

5.3.3 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</property>
    <property
name="org.osgi.framework.system.packages.extra">org.apache.log4j;version=1.2</property>
    <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 documentation.

5.3.4 Features

The current JBoss OSGi feature set in AS includes

- **Blueprint Container Support** - The OSGi Blueprint Container 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.
- **Declarative Services Support** - Declarative Services support is provided by the Apache Felix Service Component Runtime.
- **EventAdmin Support** - EventAdmin support is provided by the Apache Felix Event Admin Service.
- **Hot Deployment** - Scans the `deployments` folder for new or removed bundles.
- **HttpService and WebApp Support** - HttpService and WebApp support is provided by 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.
- **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 which provides access to a SAXParserFactory and DocumentBuilderFactory.

5.3.5 Integration Examples

The **JavaEEIntegrationTestCase** deploys 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.

You can run the examples against AS7 like any other example by specifying a target container.

```
mvn -Dtarget.container=jboss711 clean test
```

```
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", 2000);
   assertEquals("Calling PaymentService: Charged $100.0 to account 'kermit'", response);

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

   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.getName(),
null) {

           @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) : null);
    }
}
```

5.3.6 Compatibility Matrix

	JBoss-AS-7.1.0	JBoss-AS-7.1.1
JBOSGi-1.1.0	yes	
JBOSGi-1.1.1		yes

5.3.7 Command Line Interface Console

The JBoss OSGi Framework in JBoss AS 7 can be controlled via the AS 7 Command Line Interface (CLI).

The CLI supports an interactive mode as well as a scripted mode, for more information on using and launching the CLI, see the Command Line Interface documentation.

Supported operations:

- Framework Configuration
  - Setting the Subsystem Activation Mode
  - Framework Properties
  - Capabilities
- Framework Management
  - Activate the framework
  - Manage the Start Level
- Bundle Management
  - Deploying and Un-deploying
  - Starting, Stopping and Inspecting Bundles
  - Listing Bundles
- Configuration Admin
  - Adding Configuration
  - Listing Configurations
  - Inspecting Configuration
  - Removing Configuration

Framework Configuration

Framework configuration is stored in the AS 7 XML configuration file (e.g. `standalone.xml`). The CLI provides a management interface to this configuration.

Setting the Subsystem Activation Mode

The activation mode of the OSGi subsystem within AS 7 is specified in the `activation` attribute. It specifies whether the OSGi subsystem is activated on startup of the Application Server (eager) or only once the first bundle is deployed (lazy). By default the activation mode is set to `lazy`.

To read the current activation mode:

```
[standalone@localhost:9999 /] /subsystem=osgi:read-attribute(name=activation)
{
    "outcome" => "success",
    "result" => "lazy"
}
```

To change the activation mode to `eager`

```
[standalone@localhost:9999 /] /subsystem=osgi:write-attribute(name=activation,value=eager)
{"outcome" => "success"}
```

Framework Properties

Framework properties are stored as resources in the `/subsystem=osgi/property` location. Note that any changes to OSGi framework properties require a framework restart in order to become effective.

Adding a property: the following command adds the `org.osgi.framework.system.packages.extra` framework property with value `org.foo.bar;version=1.2`.

```
[standalone@localhost:9999 /]
/subsystem=osgi/property=org.osgi.framework.system.packages.extra:add(value=org.foo.bar;version=1.

{"outcome" => "success"}
```

To list the framework properties use the `ls` command:

```
[standalone@localhost:9999 /] ls /subsystem=osgi/property
org.osgi.framework.startlevel.beginning
org.osgi.framework.system.packages.extra
```

To read the value of an individual framework property:

```
[standalone@localhost:9999 /]
/subsystem=osgi/property=org.osgi.framework.startlevel.beginning:read-resource
{
    "outcome" => "success",
    "result" => {"value" => "1"}
}
```

To remove a framework property:

```
[standalone@localhost:9999 /]
/subsystem=osgi/property=org.osgi.framework.system.packages.extra:remove
{"outcome" => "success"}
```

Capabilities

The capability name is an abstract identifier for some functional capability. We currently support module identifiers (i.e. a pointer to some bundle or module available in the AS 7 product repository) and maven coordinates (i.e. a maven artifact identifier availbale in the public jboss nexus repository). Capabilities are automatically loaded when the OSGi framework is launched. In the case of bundle capabilities, these can also be started as part of this process. The modules repository can be found in the `modules/` subdirectory of the AS 7 installation, while the bundles repository can be found in the `bundles/` subdirectory.

The example command below adds a bundle capability `org.projectodd.stilts`. The system will look for capabilities at startup in the `modules/` and `bundles/` directories. In this example the org.projectodd.stilts module/bundle jar is searched in the `modules/org/projectodd/stilts/main/` and `bundles/org/projectodd/stilts/main/` locations.

```
[standalone@localhost:9999 /] /subsystem=osgi/capability=org.projectodd.stilts:add(startlevel=2)
{"outcome" => "success"}
```

Note that only bundles can be started, so specifying the `startlevel` as is done in this example does not apply to modules.

To list configured capabilities use the `ls` command:

```
[standalone@localhost:9999 /] ls /subsystem=osgi/capability
javax.servlet.api:v25           javax.transaction.api
org.apache.felix.configadmin    org.apache.felix.log
org.jboss.as.osgi.configadmin   org.jboss.osgi.logging
org.projectodd.stilts
```

To inspect a capability, use the `read-resource` command:

```
[standalone@localhost:9999 /] /subsystem=osgi/capability=org.projectodd.stilts:read-resource
{
    "outcome" => "success",
    "result" => {"startlevel" => 2}
}
```

Removing a capability can be done with the `remove` command:

```
[standalone@localhost:9999 /] /subsystem=osgi/capability=org.projectodd.stilts:remove
{"outcome" => "success"}
```

Framework Management

Framework management refers to the monitoring and manipulating the OSGi Framework runtime state.

Activate the framework

The framework can be activated using the activate command.

```
[standalone@localhost:9999 /] /subsystem=osgi:activate
{"outcome" => "success"}
```

Manage the Start Level

The current framework start level can be in found in the `startlevel` attribute in the `osgi` subsystem. For example:

```
[standalone@localhost:9999 /] ls subsystem=osgi
bundle            capability        property          activation=lazy
startlevel=1
```

To change the start level, write the desired new value to the `startlevel` attribute:

```
[standalone@localhost:9999 /] /subsystem=osgi:write-attribute(name=startlevel,value=3)
{"outcome" => "success"}
```

Bundle Management

The CLI can be used to deploy, undeploy, start and stop bundles and to inspect the current bundle state.

Deploying and Un-deploying

Deployment and un-deployment of bundles is done through the regular deployment channels in AS 7. To deploy a bundle:

```
[standalone@localhost:9999 /] deploy /home/someuser/test-osgi-bundle.jar
```

Undeploying is done with the `undeploy` command:

```
[standalone@localhost:9999 /] undeploy test-osgi-bundle.jar
```

Note that deploying a bundle will place it in the INSTALLED state, to become functional OSGi bundles need to be started.

Starting, Stopping and Inspecting Bundles

To start the `test-osgi-bundle.jar` as deployed above, use the following command:

```
[standalone@localhost:9999 /] /subsystem=osgi/bundle=test-osgi-bundle.jar:start
{"outcome" => "success"}
```

To inspect the bundle state:

```
[standalone@localhost:9999 /]
/subsystem=osgi/bundle=test-osgi-bundle.jar:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "id" => 8L,
        "location" => "test-osgi-bundle.jar",
        "startlevel" => 1,
        "state" => "ACTIVE",
        "symbolic-name" => "TestOSGiBundle",
        "type" => "bundle",
        "version" => "1.0.0"
    }
}
```

To stop the bundle, issue one of the following commands:

```
[standalone@localhost:9999 /] /subsystem=osgi/bundle=8:stop
{"outcome" => "success"}
or
[standalone@localhost:9999 /] /subsystem=osgi/bundle=test-osgi-bundle.jar:stop
{"outcome" => "success"}
```

Listing Bundles

Bundles can be listed by bundle ID using the `ls` command:

```
[standalone@localhost:9999 /] ls /subsystem=osgi/bundle
0   1   2   3   4   5   6   8
```

or in more detail using the `read-children-resources` command:

```
[standalone@localhost:9999 /]
/subsystem=osgi:read-children-resources(child-type=bundle,include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "0" => {
            "id" => 0L,
            "location" => "System Bundle",
            "startlevel" => 0,
            "state" => "ACTIVE",
            "symbolic-name" => "system.bundle",
            "type" => "bundle",
            "version" => "0.0.0"
        },
        "1" => {
            "id" => 1L,
...and so on...
```

Configuration Admin

OSGi Configuration Admin support is provided through the `configadmin` subsystem. By default the JBoss OSGi runtime in AS 7 provides the Configuration Admin Service. For more information on the OSGi Configuration Admin Service see chapter 104 in the OSGi Compendium Specification.

Configuration Admin is a dynamic system that supports re-configuration during operation, therefore all the related management operations take effect immediately and value changes will be propagated to the relevant configuration consumers without the need for a restart.

Adding Configuration

Configuration Admin configurations objects are identified by a Persistent Identified (PID) and carry an associated map of configuration values. Create a `configuration` using the `add` command:

```
[standalone@localhost:9999 /]
/subsystem=configadmin/configuration=org.example.myPid:add(entries={"key"=>"value","anotherkey"=>"
value"})
{"outcome" => "success"}
```

Listing Configurations

List all the available configurations with the `ls` command:

```
[standalone@localhost:9999 /] ls /subsystem=configadmin/configuration
org.example.myPid
```

Inspecting Configuration

Read the configuration data with the `read-resource` command:

```
[standalone@localhost:9999 /]
/subsystem=configadmin/configuration=org.example.myPid:read-resource
{
    "outcome" => "success",
    "result" => {"entries" => {
        "key" => "value",
        "anotherkey" => "another value"
    }}
}
```

Removing Configuration

Configuration objects can be removed by issuing the `remove` command on the configuration pid resource:

```
[standalone@localhost:9999 /] /subsystem=configadmin/configuration=org.example.myPid:remove

{"outcome" => "success"}
```

5.3.8 Management Console

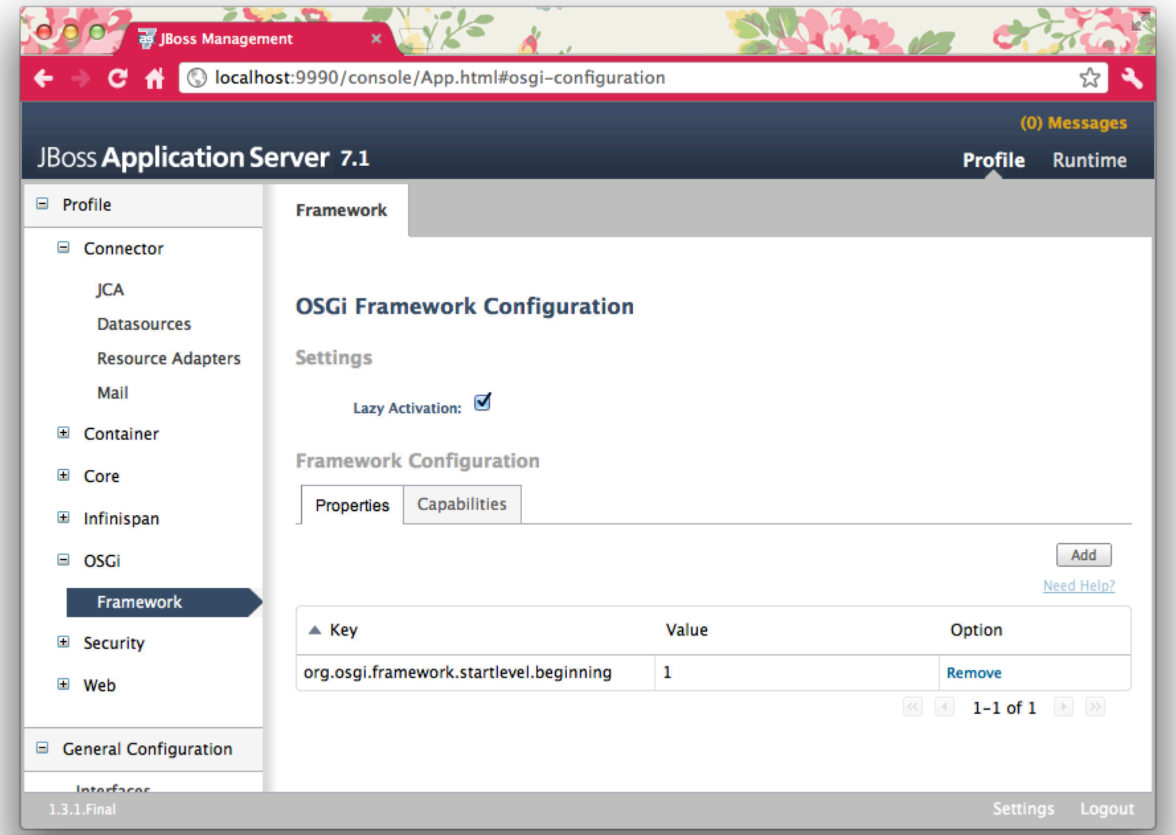
JBoss Application Server 7 comes with a web-based management console. The console can be launched by opening a browser at http://localhost:9990/console.

Supported operations:

- Framework Configuration
- Framework Management
- Bundle Management
- Configuration Admin

Framework Configuration

The OSGi framework can be configured via the **OSGi -> Framework** panels on the **Profile** section.



OSGi Framework Configuration

Settings

Lazy Activation: ✓

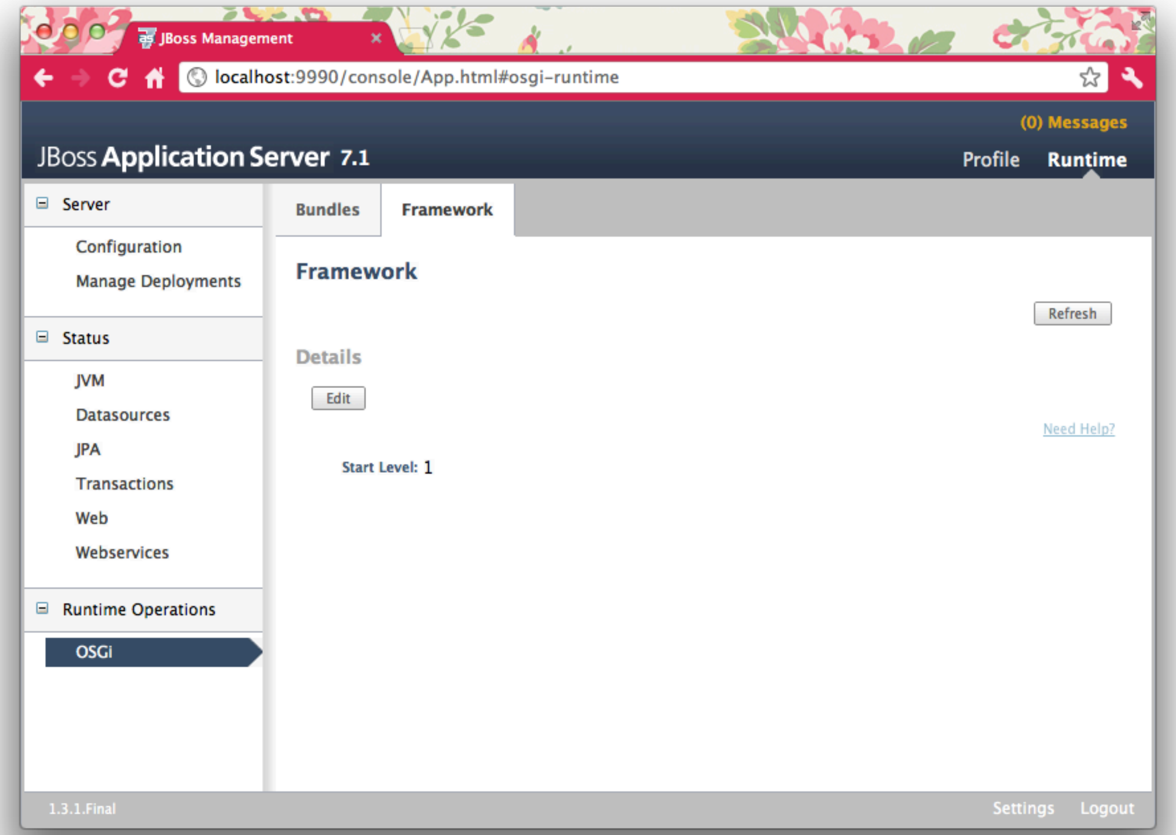
Framework Configuration

Key	Value	Option
org.osgi.framework.startlevel.beginning	1	Remove

1-1 of 1

Framework Management

The OSGi Framework can be activated through the **Runtime Operations -> OSGi -> Framework** panel in the **Runtime** section. Additionally, the current framework start level can be managed here.

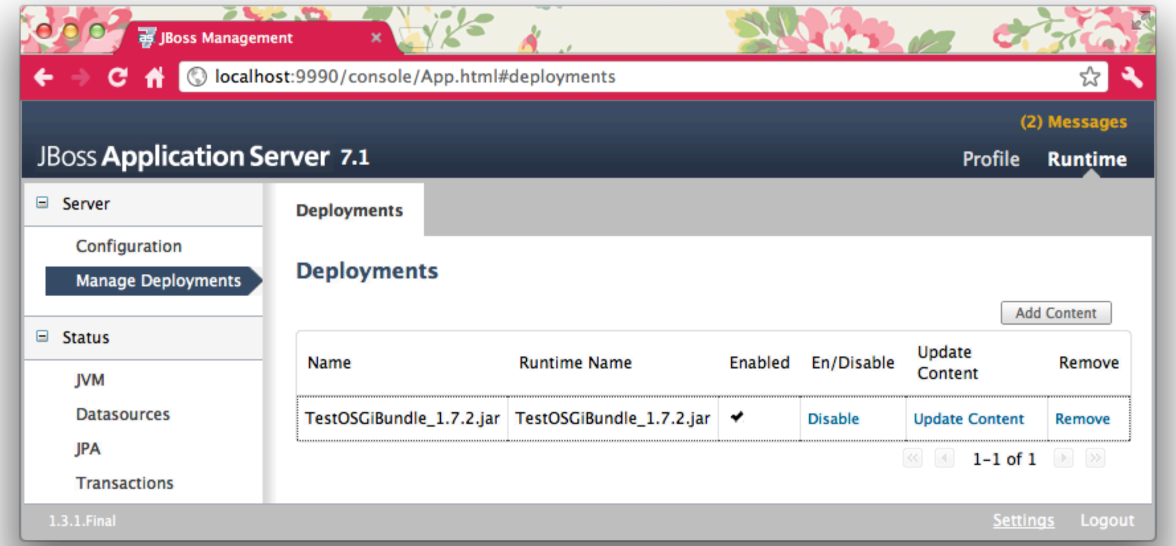


Framework

Start Level: 1

Bundle Management

Bundles can be deployed through the general AS 7 Management Console Deployment mechanism.

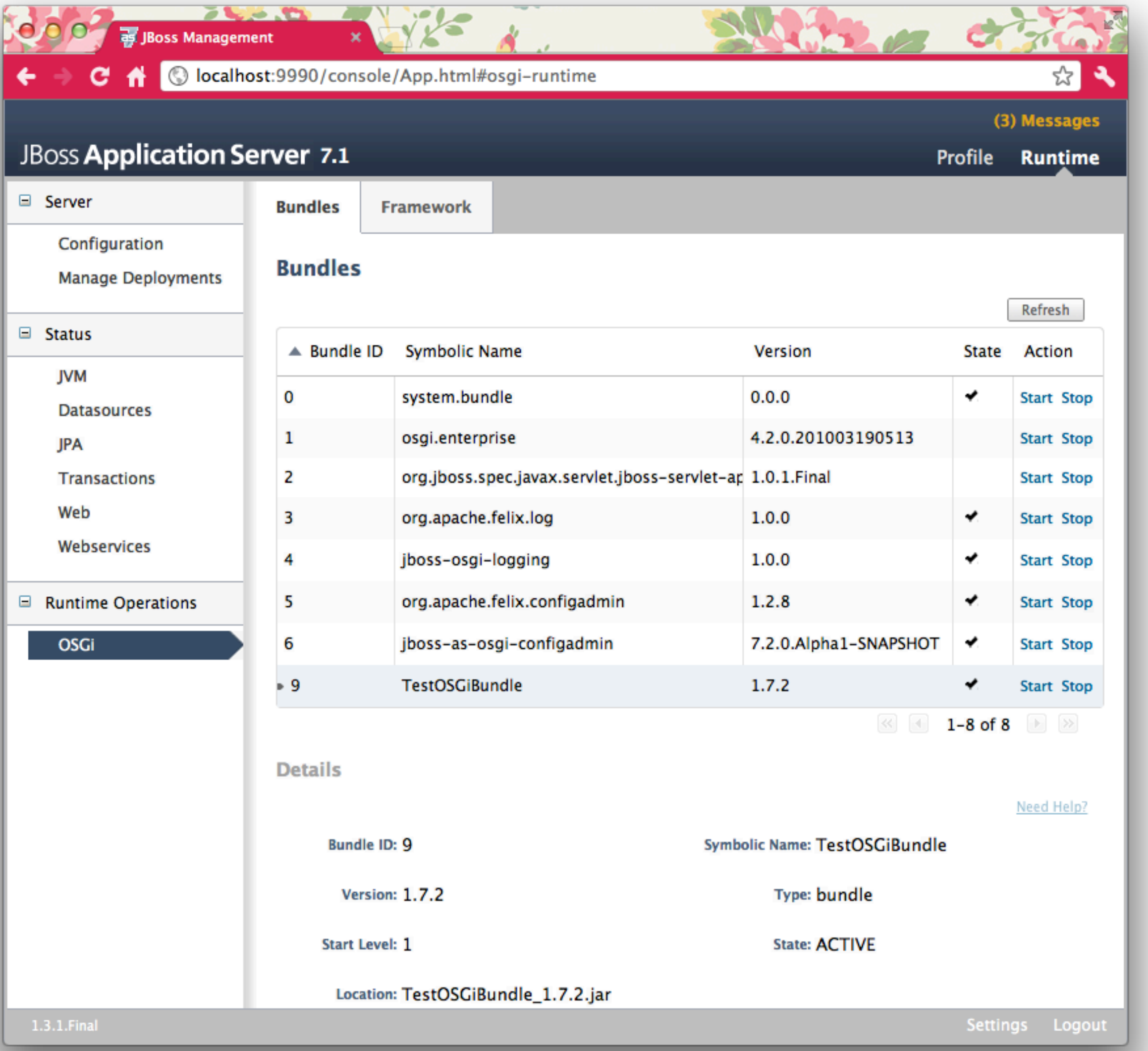


Deployments

Name	Runtime Name	Enabled	En/Disable	Update Content	Remove
TestOSGiBundle_1.7.2.jar	TestOSGiBundle_1.7.2.jar	✓	Disable	Update Content	Remove

1–1 of 1

After deploying and enabling the deployment, the bundle needs to be started, which can be done from the **Runtime Operations -> OSGi -> Bundles** panel in the **Runtime** section. Other information about bundles deployed in the Framework is available here too.



Bundles

Bundle ID	Symbolic Name	Version	State	Action
0	system.bundle	0.0.0	✓	Start Stop
1	osgi.enterprise	4.2.0.201003190513		Start Stop
2	org.jboss.spec.javax.servlet.jboss-servlet-ap	1.0.1.Final		Start Stop
3	org.apache.felix.log	1.0.0	✓	Start Stop
4	jboss-osgi-logging	1.0.0	✓	Start Stop
5	org.apache.felix.configadmin	1.2.8	✓	Start Stop
6	jboss-as-osgi-configadmin	7.2.0.Alpha1-SNAPSHOT	✓	Start Stop
9	TestOSGiBundle	1.7.2	✓	Start Stop

Details

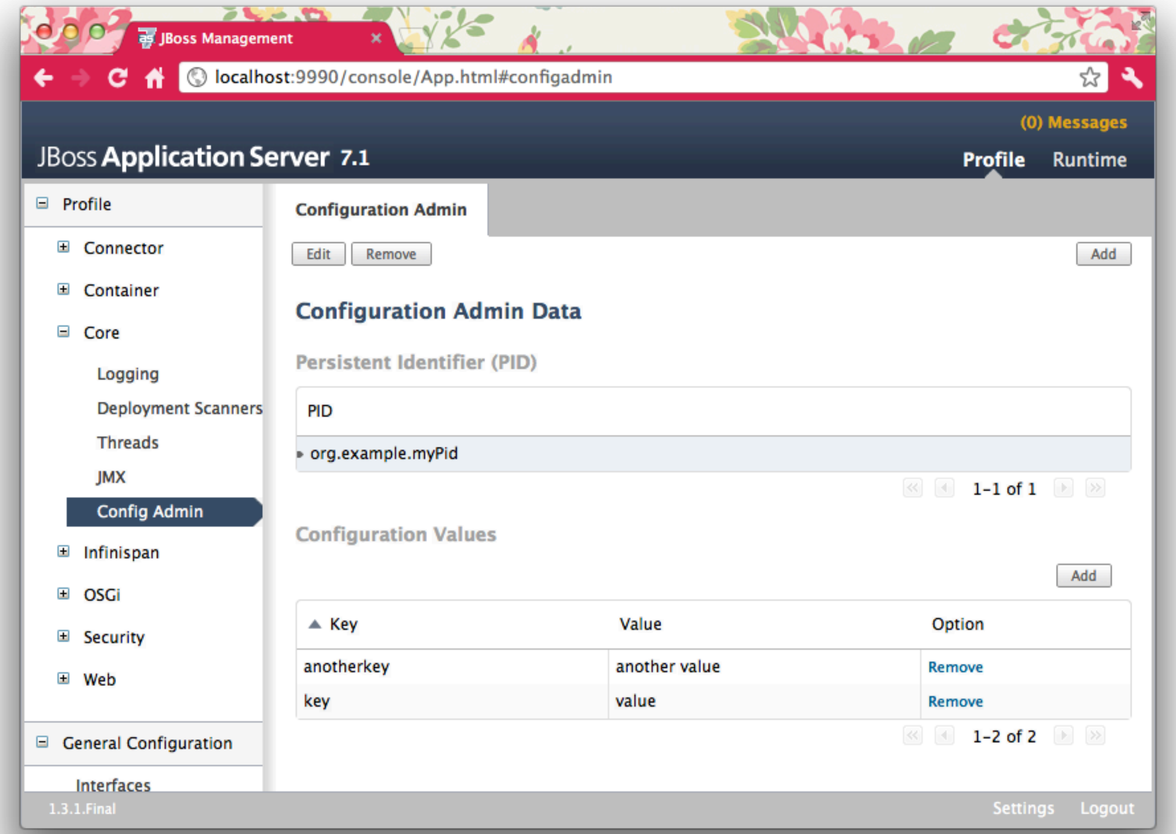
Bundle ID: 9
Symbolic Name: TestOSGiBundle
Version: 1.7.2
Type: bundle
Start Level: 1
State: ACTIVE
Location: TestOSGiBundle_1.7.2.jar

Configuration Admin

The JBoss OSGi runtime in AS 7 provides the Configuration Admin Service. For more information on the OSGi Configuration Admin Service see chapter 104 in the OSGi Compendium Specification.

Configuration Admin is a dynamic system that supports re-configuration during operation, therefore all the related management operations take effect immediately and value changes will be propagated to the relevant configuration consumers without the need for a restart.

The Configuration Admin subsystem can be managed through the **Core -> Config Admin** panel in the **Profile** section.



Configuration Admin Data

Persistent Identifier (PID)

PID
org.example.myPid

1-1 of 1

Configuration Values

Key	Value	Option
anotherkey	another value	Remove
key	value	Remove

1-2 of 2

5.4 Developer Documentation

5.4.1 Content

- Bootstrapping JBoss OSGi
- Management View
- Writing Test Cases

5.4.2 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

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

5.4.3 Management View

JBoss OSGi provides standard org.osgi.jmx management. Additional to that we provide an MBeanServer 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>
```

5.4.4 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).

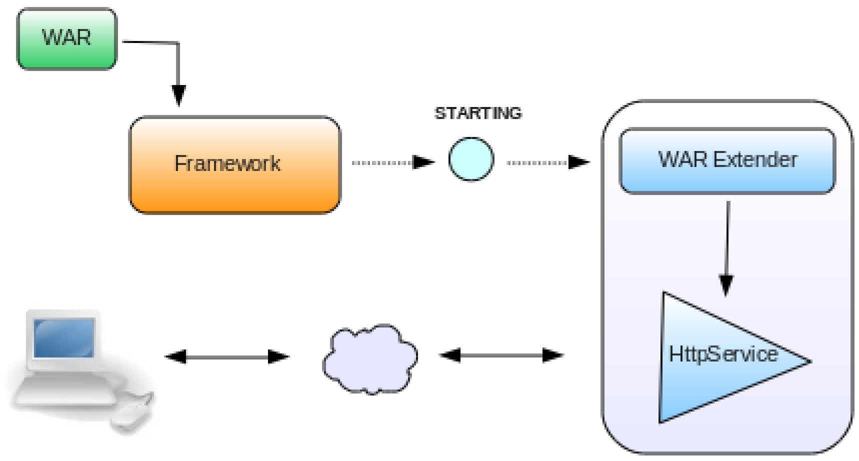
Lifecycle Interceptors

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 is often being used. JBoss OSGi profferes a differnet approach to address this problem which is covered by the Extender Pattern vs. Lifecycle Interceptor post in the JBoss OSGi Diary.

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.



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).

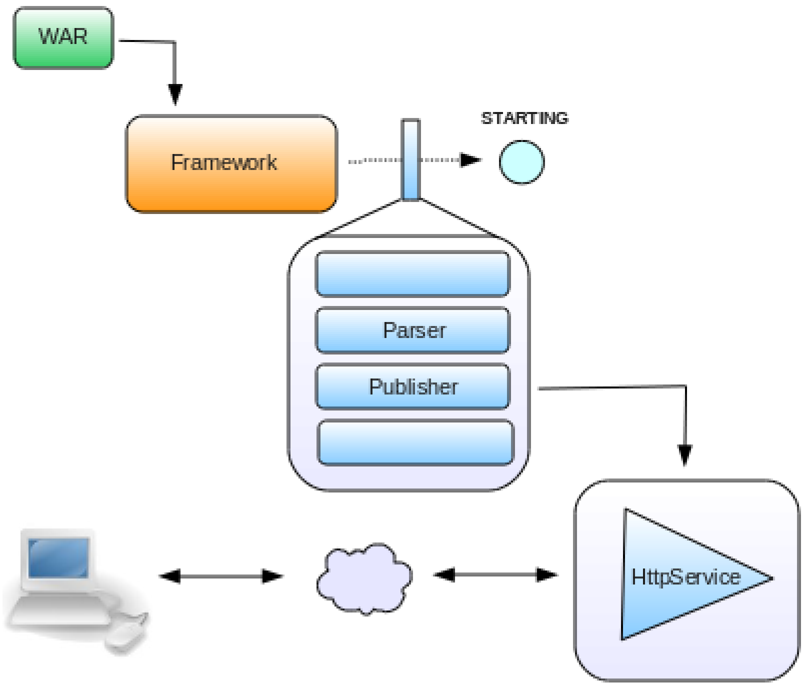
As a consequence the extender pattern is useful in some cases but not all. It is mainly useful if a bundle can optionally be extended in the true sense of the word.

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.



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.

5.5 Arquillian Test Framework

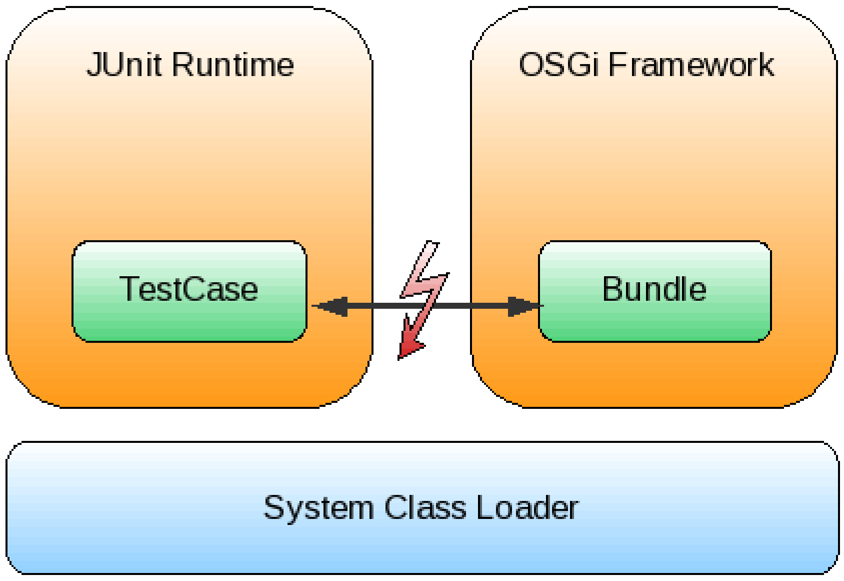
5.5.1 Content

5.5.2 Overview

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.



For this release, we extended the Arquillian Test Framework 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

5.5.3 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 to communicate between the test client and the remote OSGi Framework.

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

5.5.4 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-arquillian");
      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.getName());
      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());
   }
}
```

5.6 Provided Examples

5.6.1 Content

5.6.2 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.

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
```

5.6.3 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.blueprint.bundle.ServiceA">
  </service>

  <service id="serviceB" interface="org.jboss.test.osgi.example.blueprint.bundle.ServiceB">
    <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"/>
```

5.6.4 Configuration Admin

The **ConfigurationAdminTestCase** shows how an OSGi ManagedService can be configured through the ConfigurationAdmin service.

```
public void testManagedService() throws Exception {

   // Get the {@link Configuration} for the given PID
   Configuration config = configAdmin.getConfiguration(ConfiguredService.SERVICE_PID);
   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.class.getName());
   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"/>
```

5.6.5 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(), null) {
     public Object addingService(ServiceReference reference) {
         Comparator<Object> service = (Comparator<Object>) super.addingService(reference);
         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"/>
```

5.6.6 Event Admin

The **EventAdminTestCase** uses the EventAdmin 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"/>
```

5.6.7 HttpService

The **HttpServiceTestCase** deploys a Service that registeres a servlet and a resource with the HttpService.

```
ServiceTracker tracker = new ServiceTracker(context, HttpService.class.getName(), null);
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, null);
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 these capabilities

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

5.6.8 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.getName(), null)
      {
         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.getName());
      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,
BundleStateMBean.class);
   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.

5.6.9 JTA Service

The **TransactionTestCase** gets the UserTransaction service and registers a transactional user object (i.e. one that implements Synchronization) with the TransactionManager 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.getName());
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.getName());
   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"/>
```

5.6.10 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.

```
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, null);
      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.

5.6.11 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</servlet-class>
    <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" startlevel="1"/>
<capability name="org.ops4j.pax.web:pax-web-jsp:1.1.2" startlevel="1"/>
<capability name="org.ops4j.pax.web:pax-web-extender-war:1.1.2" startlevel="1"/>
```

5.6.12 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(context,
bundle);
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, bundle);
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"/>
```

5.7 References

5.7.1 Resources

- OSGi 4.2 Specifications
- OSGi 4.2 JavaDoc
- OSGi on Wikipedia
- JBoss OSGi Project Page
- JBoss OSGi Wiki
- JBoss OSGi Diary
- Issue Tracking
- Source Repository
- User Forum
- Design Forum

5.7.2 Authors

- Thomas Diesler
- David Bosschaert

5.8 Getting Support

We offer free support through the JBoss OSGi User Forum.

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.

6 Developer Guide

6.1 Project Infrastructure

6.1.1 Downloads

JBoss OSGi can be downloaded as part of JBoss Application Server 7.

The Standalone Runtime can be downloaded from SourceForge.

6.1.2 Forums

- JBossOSGi User Forum
- JBossOSGi Development Forum

6.1.3 Mailing Lists

- jboss-osgi-announce
- jboss-osgi-issues
- jboss-osgi-users
- jboss-osgi-dev

6.1.4 IRC

#jbosgi @ irc.freenode.net

6.1.5 GitHub hosted SCM

The JBoss OSGi umbrella project lives here: http://github.com/jbosgi/jbosgi

6.1.6 Issue Tracker

http://jira.jboss.com/jira/browse/JBOSGI

6.1.7 Continous Hudson QA

https://hudson.qa.jboss.com/hudson/view/JBossOSGi

6.2 Building from Source

All components used in the JBoss OSGi project have their source code hosted at github and are built using Maven 3.

This page describes the building process for the umbrella project, the project that pulls all the subcomponents together, builds and tests JBoss OSGi. Except where explicitly stated, the procedure outlined on this page applies to all other JBoss OSGi components.

6.2.1 Clone from GitHub

The following command clones the umbrella project into your workspace:

```
git clone git://github.com/jbosgi/jbosgi.git
```

Umbrella only: When you work with the latest umbrella master at git://github.com/jbosgi/jbosgi.git it is likely that the project contains references to external submodules, in which case you have a .gitmodules in the project root. Before you do the actual maven build, you need to initialize and update the submodules like this:

```
git submodule init
git submodule update
```

6.2.2 Configuring Maven to use the JBoss Repository

To use dependencies from the jboss.org repository the following snippet should be included in settings.xml

```
<settings>
  <profiles>
    <profile>
      <id>jboss-public-repository</id>
      <repositories>
        <repository>
          <id>jboss-public-repository-group</id>
          <name>JBoss Public Maven Repository Group</name>
          <url>https://repository.jboss.org/nexus/content/groups/public/</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-public-repository-group</id>
          <name>JBoss Public Maven Repository Group</name>
          <url>https://repository.jboss.org/nexus/content/groups/public/</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <activeProfiles>
    <activeProfile>jboss-public-repository</activeProfile>
  </activeProfiles>

</settings>
```

For more details, see Configuring Maven

6.2.3 Build with Maven

The command below builds the project and runs the suites:

```
[tdiesler@tdvaio trunk]$ mvn clean install
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] JBossOSGi ............................................ SUCCESS [0.071s]
[INFO] JBossOSGi Reactor .................................... SUCCESS [0.067s]
[INFO] JBossOSGi Testsuite .................................. SUCCESS [1.095s]
[INFO] JBossOSGi Testsuite - Examples ....................... SUCCESS [1:35.635s]
[INFO] JBossOSGi Testsuite - Functional ..................... SUCCESS [1:03.949s]
[INFO] JBossOSGi Testsuite - Performance .................... SUCCESS [53.991s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4 minutes 51 seconds
[INFO] Finished at: Thu Mar 04 08:21:04 CET 2010
[INFO] Final Memory: 74M/148M
[INFO] ------------------------------------------------------------------------
```

6.3 Building the Installer

JBossOSGi uses the IzPack installer for packaging the distribution.

```
[tdiesler@tdvaio jbosgi]$ mvn -Pdistro install
[INFO] Scanning for projects...
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] JBossOSGi
[INFO] JBossOSGi Reactor
[INFO] JBossOSGi Testsuite
[INFO] JBossOSGi Testsuite Examples
[INFO] JBossOSGi Testsuite Functional
[INFO] JBossOSGi Testsuite Performance
[INFO] JBossOSGi Distribution
[INFO] JBossOSGi Distribution Javadoc
[INFO] JBossOSGi Distribution Installer
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 54.796s
[INFO] Finished at: Mon Jul 25 17:44:30 CEST 2011
[INFO] Final Memory: 28M/147M
[INFO] ------------------------------------------------------------------------
```

Finally, run the generated install jar to obtain the set of documentation and OSGi examples that can get executed against the embedded Framework or one of the supported remote target containers (i.e. AS7).

```
java -jar distribution/installer/target/jboss-osgi-installer-1.0.0.jar
```

6.4 Javadoc Guidelines

6.4.1 General

- Document the background knowledge, the intention and not the result
- You should write **what** the class/method does, **not how** it does it.
- {@link MyClass#someMethod} should be written the first time someMethod is mentioned in the text. Subsequent mentions should not have links.
- Generally use @link, @see and @seealso generously.

6.4.2 Interface description must consist of

- Purpose
- Description (optional)

6.4.3 Class description must consist of

- Purpose
- Description (optional)
- Statement whether this class implements an entity or a process defined by a standard specification
- Statement whether instances of this class are mutable (optional).
- Statement whether this class is multi-threading safe (optional).

6.4.4 Method description must consist of

- Purpose
- Expected effect
- Description (optional)
- Statement whether this method is modal (optional).
- Required parameters. It must be explicitly stated whether null is permitted as a parameter value.
- Per default parameters are assumed to require a non-null value and to cause a IllegalArgumentException if null is given.
- Exceptions that can be thrown in the course of method's execution and their possible cause (post-conditions).
- Private methods can have minimal or no JavaDocs if they are short and trivial enough.
- Methods that implement a signature from an interface or base class that is sufficiently documented there can omit JavaDocs

6.5 Release Cycle

JBoss OSGi has a release cycle of eight weeks per minor release.

To improve the predictability of releases, the following rules apply:

6.5.1 The Release Cycle

```
---------------------------------
W1
W2 Define set of JIRA issues
----------------------------------
W3
W4
W5
W6 Work on issues
----------------------------------
W7
W8 QA, Docs, Packaging
----------------------------------
```

At the end of W2 there are no more JIRA issues added to a release. Folks have agreed to resolve their respective issues for the given release cycle. There are also no more unassigned issues after W2. Generally, issues are assigned to the person who is supposed to make the next progress step. To minimize conflicts due to overlapping work, progress is indicated using the Start/Stop Progress feature in Jira.

Development continues in trunk of the respective projects until we reach code freeze at the end of W6. At the end of W6 there should only be release or documentation related issues left unresolved.

Then we branch and do the remaining QA related work in that branches. Finally we tag from the QA branches.

6.6 Release Procedure

6.6.1 Publish snapshots of the API & Userguide

Before you can start, make sure that your public SSH key is registered for use with osgi@filemgmt.jboss.org. This key will be used instead of a password. You can do this by emailing the public key to help@jboss.org.

Obtain a snapshot of the API docs either by building http://github.com/jbosgi/jbosgi/tree/master/distribution or by downloading them from a successful hudson build. The API docs are built at distribution/javadoc/target/apidocs.

Then copy them to a specific location for this release in the documentation directory using scp, e.g. for jboss-osgi-1.0.0:

```
cd distribution/javadoc/target
mkdir jboss-osgi-1.0.0; mv apidocs jboss-osgi-1.0.0
scp -r jboss-osgi-1.0.0 osgi@filemgmt.jboss.org:docs_htdocs/osgi/jboss-osgi-1.0.0
```

Then update the link from the API docs to the latest versions:

```
$ sftp osgi@filemgmt.jboss.org:docs_htdocs/osgi
Connecting to filemgmt.jboss.org...
Changing to: /docs_htdocs/osgi
sftp> rm apidocs
sftp> ln -s jboss-osgi-1.0.0/apidocs apidocs
```

Check that you see the desired results at:

- http://docs.jboss.org/osgi/apidocs

6.6.2 Update Release Documents

- Copy the release notes from the JIRA Roadmap to ChangeLog.html
- Extract the relevant changes to ReleaseNotes.html
- Add the user and developer guides to the distribution dir

6.6.3 Tag and Deploy

Do the release using the Maven Release Plugin

```
>mvn -Pdistro release:prepare
>mvn -Pdistro release:perform
```

Close and promote the release in the Nexus Repository.
Follow Deploying a Release on how to do this.

6.6.4 Publish the artifacts on SourceForge

Upload the installer, the changelog and the release notes.

Check the result at the JBossOSGi download area

6.6.5 Advertise the release

- Create a news wiki page and link to it from the main wiki page
- Blog about the release on JBossOSGi Diary (http://jbossosgi.blogspot.com)
- Advertise the release on mailing lists (i.e. jboss-development@lists.jboss.org, thecore@jboss.org)

6.7 OSGi TCK Setup

6.7.1 Access Restrictions

There is **no public access** to the OSGi TCK.

You either need to have an OSGi Alliance member login or a RedHat Kerberos Login.
This article is related to JBOSGI-156

6.7.2 Getting the OSGi TCK

The easiest way to get the OSGi TCK is to do an SVN checkout

```
svn co https://svn.devel.redhat.com/repos/jboss-tck/osgitck/r4v42
```

For this you'll need to be connected to the VPN and use your Kerberos login.

You can get the original from here: https://www.osgi.org/members/svn/build/tags/r4v42-core-cmpn-final

6.7.3 Setup the OSGi TCK

Checkout and build the JBOSGi Framework

```
git clone git://github.com/jbosgi/jbosgi-framework.git
cd jbosgi-framework
mvn -Pall clean install
```

Copy and edit the setup properties

```
cd tcksetup
cp ant.properties.example ant.properties
vi ant.properties
```

Running the OSGi TCK against the RI (Equinox)

```
ant setup.ri
ant run-core-tests
ant test-reports
```

Running the OSGi TCK against the JBoss OSGi Framework

```
ant setup.vi
ant run-core-tests
ant test-reports
```

Other target are:

```
clean                   Clean the TCK setup
run-blueprint-tests     Run the TCK blueprint tests
run-core-tests          Run the TCK core tests
run-jdbc-tests          Run the TCK jdbc tests
run-jmx-tests           Run the TCK jmx tests
run-jndi-tests          Run the TCK jndi tests
run-jpa-tests           Run the TCK jpa tests
run-jta-tests           Run the TCK jta tests
run-packageadmin-tests  Run the TCK Package Admin service tests
run-startlevel-tests    Run the TCK Start Level service tests
run-webapp-tests        Run the TCK webapp tests
setup.ri                Setup the TCK using the RI (Equinox)
setup.vi                Setup the TCK using the Vendor Implemenation
test-reports            Generate the TCK test reports
update-framework        Update the JBoss OSGi Framework
```

Default target: setup.vi

6.7.4 Installing the Eclipse Bnd plugin

Eclipse uses the aQute Bnd plugin to setup the classpath container and for JUnit integration.

Copy the bnd-356.jar to your Eclipse plugins directory.

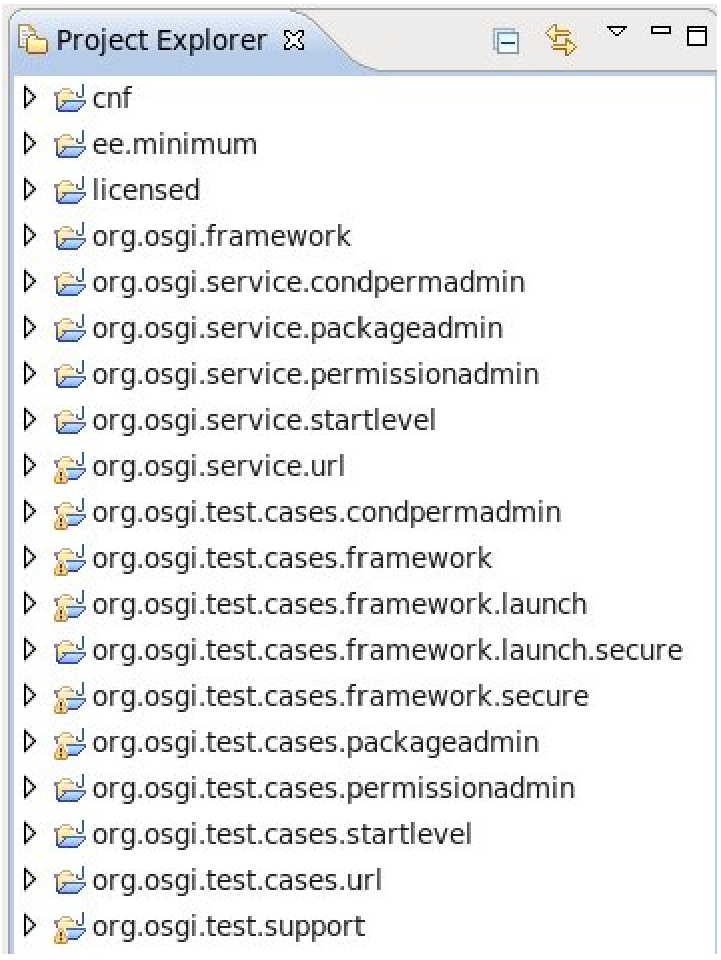
Please note, the TCK setup uses a bnd version that supports standard JUnit test reports (0.0.365.SP1). This version however, does not work properly as an Eclipse plugin.

6.7.5 Importing the TCK projects

The layout for the core tests is defined in osgi.ct/layout.bnd.

```
build.core.tests =
    org.osgi.test.cases.framework,
    org.osgi.test.cases.framework.secure,
    org.osgi.test.cases.framework.launch,
    org.osgi.test.cases.framework.launch.secure,
    org.osgi.test.cases.packageadmin,
    org.osgi.test.cases.condpermadmin,
    org.osgi.test.cases.permissionadmin,
    org.osgi.test.cases.startlevel,
    org.osgi.test.cases.url
```

Additional to the core test projects, you need to import various other support projects.

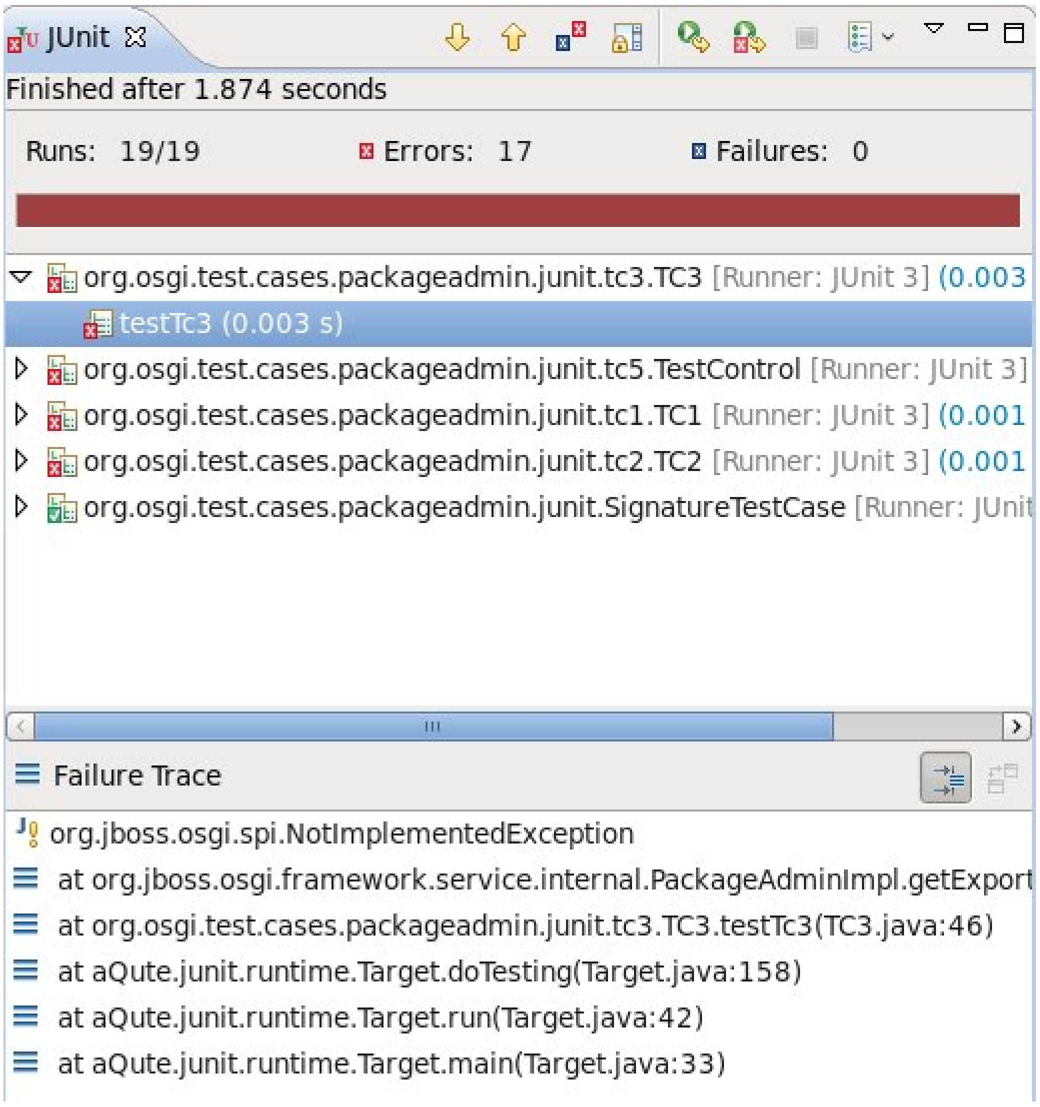


- cnf
- ee.minimum
- licensed
- org.osgi.framework
- org.osgi.service.condpermadmin
- org.osgi.service.packageadmin
- org.osgi.service.permissionadmin
- org.osgi.service.startlevel
- org.osgi.service.url
- org.osgi.test.cases.condpermadmin
- org.osgi.test.cases.framework
- org.osgi.test.cases.framework.launch
- org.osgi.test.cases.framework.launch.secure
- org.osgi.test.cases.framework.secure
- org.osgi.test.cases.packageadmin
- org.osgi.test.cases.permissionadmin
- org.osgi.test.cases.startlevel
- org.osgi.test.cases.url
- org.osgi.test.support

Additional to that you would also want to import the projects that you want to test (i.e. framework)

6.7.6 Running TCK tests as JUnit tests

Right-click on the test project you want to run and select "Run As|OSGi JUnit". The Bnd output should appear in the console window.



```
Finished after 1.874 seconds

Runs:  19/19          Errors:  17          Failures:  0

org.osgi.test.cases.packageadmin.junit.tc3.TC3 [Runner: JUnit 3] (0.003
    testTc3 (0.003 s)
org.osgi.test.cases.packageadmin.junit.tc5.TestControl [Runner: JUnit 3]
org.osgi.test.cases.packageadmin.junit.tc1.TC1 [Runner: JUnit 3] (0.001
org.osgi.test.cases.packageadmin.junit.tc2.TC2 [Runner: JUnit 3] (0.001
org.osgi.test.cases.packageadmin.junit.SignatureTestCase [Runner: JUnit

Failure Trace
org.jboss.osgi.spi.NotImplementedException
 at org.jboss.osgi.framework.service.internal.PackageAdminImpl.getExport
 at org.osgi.test.cases.packageadmin.junit.tc3.TC3.testTc3(TC3.java:46)
 at aQute.junit.runtime.Target.doTesting(Target.java:158)
 at aQute.junit.runtime.Target.run(Target.java:42)
 at aQute.junit.runtime.Target.main(Target.java:33)
```

6.7.7 Running an individual TCK test

When you have sucessfully run an entire test section you can right-click on an individual test and run/debug it seperately. Breakpoints should work.

Good Luck

6.7.8 R4v42 TCK Setup

Access Restrictions

There is **no public access** to the OSGi TCK.

You either need to have an OSGi Alliance member login or a RedHat Kerberos Login.
This article is related to JBOSGI-156

Getting the OSGi TCK

The easiest way to get the OSGi TCK is to do an SVN checkout

```
svn co https://svn.devel.redhat.com/repos/jboss-tck/osgitck/r4v42
```

For this you'll need to be connected to the VPN and use your Kerberos login.

You can get the original from here: https://www.osgi.org/members/svn/build/tags/r4v42-core-cmpn-final

Setup the OSGi TCK

Checkout and build the JBOSGi Framework

```
git clone git://github.com/jbosgi/jbosgi-framework.git
cd jbosgi-framework
mvn -Pall clean install
```

Copy and edit the setup properties

```
cd tcksetup
cp ant.properties.example ant.properties
vi ant.properties
```

Running the OSGi TCK against the RI (Equinox)

```
ant setup.ri
ant run-core-tests
ant test-reports
```

Running the OSGi TCK against the JBoss OSGi Framework

```
ant setup.vi
ant run-core-tests
ant test-reports
```

Other target are:

```
clean                    Clean the TCK setup
run-blueprint-tests      Run the TCK blueprint tests
run-core-tests           Run the TCK core tests
run-jdbc-tests           Run the TCK jdbc tests
run-jmx-tests            Run the TCK jmx tests
run-jndi-tests           Run the TCK jndi tests
run-jpa-tests            Run the TCK jpa tests
run-jta-tests            Run the TCK jta tests
run-packageadmin-tests   Run the TCK Package Admin service tests
run-startlevel-tests     Run the TCK Start Level service tests
run-webapp-tests         Run the TCK webapp tests
setup.ri                 Setup the TCK using the RI (Equinox)
setup.vi                 Setup the TCK using the Vendor Implemenation
test-reports             Generate the TCK test reports
update-framework         Update the JBoss OSGi Framework
```

Default target: setup.vi

Installing the Eclipse Bnd plugin

Eclipse uses the aQute Bnd plugin to setup the classpath container and for JUnit integration.

Copy the bnd-356.jar to your Eclipse plugins directory.

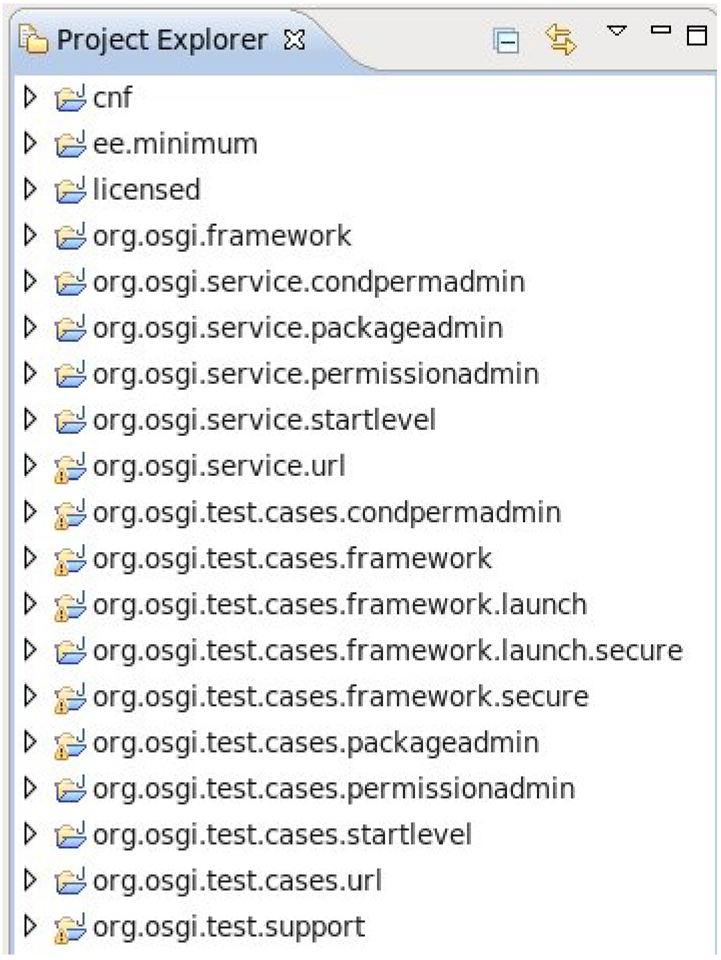
Please note, the TCK setup uses a bnd version that supports standard JUnit test reports (0.0.365.SP1). This version however, does not work properly as an Eclipse plugin.

Importing the TCK projects

The layout for the core tests is defined in osgi.ct/layout.bnd.

```
build.core.tests =
    org.osgi.test.cases.framework,
    org.osgi.test.cases.framework.secure,
    org.osgi.test.cases.framework.launch,
    org.osgi.test.cases.framework.launch.secure,
    org.osgi.test.cases.packageadmin,
    org.osgi.test.cases.condpermadmin,
    org.osgi.test.cases.permissionadmin,
    org.osgi.test.cases.startlevel,
    org.osgi.test.cases.url
```

Additional to the core test projects, you need to import various other support projects.

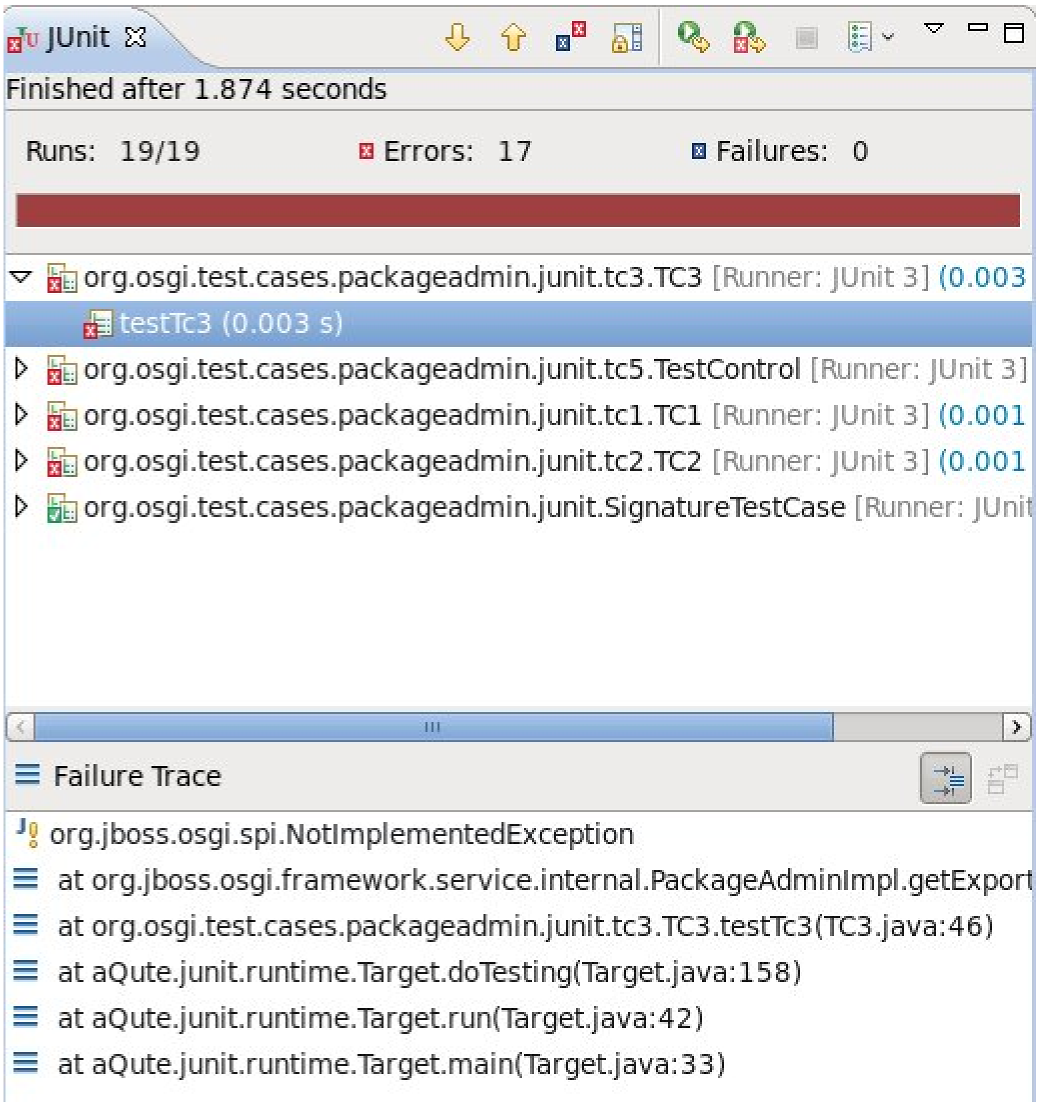

Project Explorer

cnf
ee.minimum
licensed
org.osgi.framework
org.osgi.service.condpermadmin
org.osgi.service.packageadmin
org.osgi.service.permissionadmin
org.osgi.service.startlevel
org.osgi.service.url
org.osgi.test.cases.condpermadmin
org.osgi.test.cases.framework
org.osgi.test.cases.framework.launch
org.osgi.test.cases.framework.launch.secure
org.osgi.test.cases.framework.secure
org.osgi.test.cases.packageadmin
org.osgi.test.cases.permissionadmin
org.osgi.test.cases.startlevel
org.osgi.test.cases.url
org.osgi.test.support

Additional to that you would also want to import the projects that you want to test (i.e. framework)

Running TCK tests as JUnit tests

Right-click on the test project you want to run and select "Run As|OSGi JUnit". The Bnd output should appear in the console window.



```
Finished after 1.874 seconds

Runs: 19/19          Errors: 17          Failures: 0

org.osgi.test.cases.packageadmin.junit.tc3.TC3 [Runner: JUnit 3] (0.003
    testTc3 (0.003 s)
org.osgi.test.cases.packageadmin.junit.tc5.TestControl [Runner: JUnit 3]
org.osgi.test.cases.packageadmin.junit.tc1.TC1 [Runner: JUnit 3] (0.001
org.osgi.test.cases.packageadmin.junit.tc2.TC2 [Runner: JUnit 3] (0.001
org.osgi.test.cases.packageadmin.junit.SignatureTestCase [Runner: JUnit
```

```
org.jboss.osgi.spi.NotImplementedException
 at org.jboss.osgi.framework.service.internal.PackageAdminImpl.getExport
 at org.osgi.test.cases.packageadmin.junit.tc3.TC3.testTc3(TC3.java:46)
 at aQute.junit.runtime.Target.doTesting(Target.java:158)
 at aQute.junit.runtime.Target.run(Target.java:42)
 at aQute.junit.runtime.Target.main(Target.java:33)
```

Running an individual TCK test

When you have sucessfully run an entire test section you can right-click on an individual test and run/debug it seperately. Breakpoints should work.

Good Luck

6.7.9 R5 TCK Setup

Getting the OSGi TCK

There is **no public access** to the OSGi TCK.

- Setup the OSGi TCK
- Installing the Eclipse Bnd plugin
- Importing the TCK projects
- Setup the TCK on Jenkins

Getting the TCK from RedHat internal Git

We internally host the TCK here

```
git clone git://git.app.eng.bos.redhat.com/org.osgi.tck.git
    git checkout r5-core-ri-ct-final
    ...
    HEAD is now at 8ae7389... Updated bnd and changes required to build
```

Getting the OSGi TCK form the OSGi Alliance

You need to have an OSGi Alliance member login.
The OSGi Alliance has a page on Installing and Setting up Git

```
git clone https://www.osgi.org/members/git/build org.osgi.build
    git checkout r5-core-ri-ct-final
    ...
    HEAD is now at 8ae7389... Updated bnd and changes required to build
```

Setup the OSGi TCK

Checkout and build the JBOSGi Framework

```
git clone git://github.com/jbosgi/jbosgi-framework.git
cd jbosgi-framework; git checkout r5
mvn -Pall clean install
```

Copy and edit the setup properties

```
cd tcksetup
cp ant.properties.example ant.properties
vi ant.properties
```

Running the OSGi TCK against the RI (Equinox)

```
ant setup.ri
ant run-core-tests
ant test-reports
```

Running the OSGi TCK against the JBoss OSGi Framework

```
ant setup.vi
ant run-core-tests
ant test-reports
```

Other target are:

```
clean                   Clean the TCK setup
run-core-tests          Run the TCK core tests
setup.ri                Setup the TCK using the RI (Equinox)
setup.vi                Setup the TCK using the Vendor Implemenation
test-reports            Generate the TCK test reports
update-framework        Update the JBoss OSGi Framework
```

Default target: setup.vi

Installing the Eclipse Bnd plugin

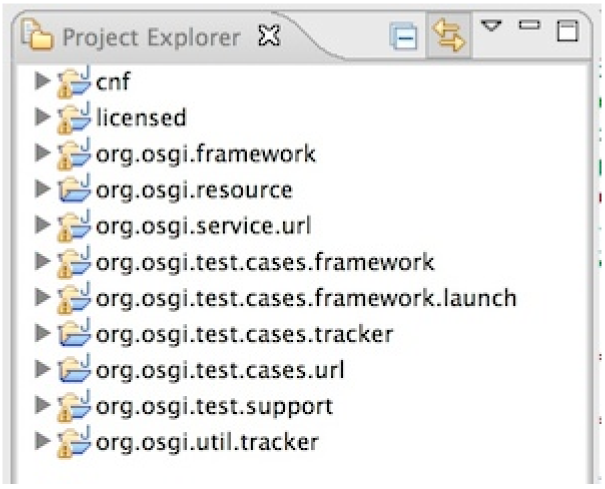
There is a mail from BJ Hargrave on bndtools @ OSGi that describes what needs to be done to make the TCK run in Eclipse

Importing the TCK projects

The layout for the core tests is defined in osgi.ct/layout.bnd.

```
build.core.tests = \
    org.osgi.test.cases.framework, \
    org.osgi.test.cases.framework.secure, \
    org.osgi.test.cases.framework.launch, \
    org.osgi.test.cases.framework.launch.secure, \
    org.osgi.test.cases.condpermadmin, \
    org.osgi.test.cases.permissionadmin, \
    org.osgi.test.cases.url, \
    org.osgi.test.cases.tracker
```

Additional to the core test projects, you need to import various other support projects.



cnf
licensed
org.osgi.framework
org.osgi.resource
org.osgi.service.url
org.osgi.test.cases.framework
org.osgi.test.cases.framework.launch
org.osgi.test.cases.tracker
org.osgi.test.cases.url
org.osgi.test.support
org.osgi.util.tracker

Additional to that you would also want to import the projects that you want to test (i.e. framework)

Setup the TCK on Jenkins

The following steps need to be done only once to setup the TCK in the static Jenkins build environment.

```
[tdiesler@tdmac ~] $ ssh dev85.mw.lab.eng.bos.redhat.com
...
[tdiesler@dev85 ~]$ sudo su - hudson
[hudson@dev85 ~]$ git clone git://git.app.eng.bos.redhat.com/org.osgi.tck.git
static_build_env/org.osgi.tck
...
[hudson@dev85 ~]$ cd static_build_env/org.osgi.tck
[hudson@dev85 org.osgi.tck]$ git tag
r4v42-enterprise-ri-ct-final
r4v43-core-ri-ct-final
r5-core-ri-ct-final
r5-enterprise-ri-ct-final
```

A Jenkins job can clone the TCK from the static build environment like this

```
export TCKCHECKOUT=$WORKSPACE/tckcheckout

# Clone the TCK repo
if [ ! -d $TCKCHECKOUT ]; then
  git clone /home/hudson/static_build_env/org.osgi.tck $TCKCHECKOUT
fi

# Switch to known tckcheckout tag
cd $TCKCHECKOUT; git checkout r5-core-ri-ct-final
```

For an example, see the osgi.core.r5v50 job.

Good Luck

6.8 JBossOSGi Logging

Logging id ranges for JBossOSGi i18n message interfaces.

Component	**Range**
jbosgi-vfs	10000-10199
jbosgi-spi	10200-10399
jbosgi-deployment	10400-10599
jbosgi-metadata	10600-10799
jbosgi-resolver	10800-10999
jbosgi-framework	11000-11399
Bundle	Range
jbosgi-jmx	20000-20199
jbosgi-logging	20200-20399
jbosgi-repository	20400-20599
jbosgi-xerces	20600-20799

6.9 Compatibillity Matrix

AS7	**7.1.0.Final**	**7.1.1.Final**	**7.1.2.Final**
jbosgi-1.1.0	ok		
jbosgi-1.1.1	ok	ok	
jbosgi-1.1.2			ok

7 News

- 18-Jun-2012 JBossOSGi-1.1.1 Released
- 08-Mar-2012 JBossOSGi-1.1.0 Released
- 26-Jul-2011 JBossOSGi-1.0.0 Released

7.1 Future Roadmap

This document outlines the JBoss OSGi Roadmap.

7.1.1 Short Term Roadmap

- JBoss OSGi 1.0.1

7.1.2 Medium Term Roadmap

- Make JNDI integration fully compliant (JBOSGI-81)
- Core 4.3 support (JBOSGI-497)
- JPA integration (JBOSGI-260)
- Migrate Web support to JBossWeb (AS7-351, AS7-1556)

7.1.3 Past releases

- JBoss OSGi 1.0 released

7.2 JBossOSGi-1.0.0

I am happy to announce the release of JBossOSGi-1.0.0.

You can download the binary here: jboss-osgi-installer-1.0.0.jar

This is our first OSGi Core 4.2. Framework compliant release. It also comes with improvements in the following areas

- Integration with JBoss AS7
- Pass the Core 4.2 Compliance Testsuite from the OSGi Alliance
- Add Declarative Services (DS) example
- Wiki documentation now available in Confluence
- OSGi service invocation from EJB3 and Webap
- Migration to the Arquillian test framework
- Better support for execution environments

For details please have a look at the latest version of our User Guide.

Here are the change log details

Enjoy

7.3 JBossOSGi-1.1.0

I am happy to announce the release of JBossOSGi-1.1.0.

You can download the binary here: jboss-osgi-installer-1.1.0.jar

This is a OSGi Core 4.2. Framework compliant release with a strong focus on AS7 integration.

It also comes with improvements in the following areas

- Concurrency optimizations throughout the framework
- Improved access to system and boot delegation packages
- Direct integration of many Felix, Aries and Ops4J bundles
- Added support for JSP pages in OSGi Web Applications
- Initial implementation of an OSGi Repository
- Capture OSGi frequently asked questions in the wiki
- Comprehensive coverage of AS7 / OSGi integration in user guide
- Add support for AS7 as remote testing target
- Replace the Standalone Runtime with AS7

Here are the change log details.

Additional to that we fixed 80+ OSGi related issues in the jboss-as-7.1.0 code base. Most notably there are

- Expose OSGi management through domain API
- Provide injectable BundleContext in JNDI
- Transactional integration endpoint (STOMP)
- Provide Repository API as general MSC service
- Remove Felix, Aries and Ops4J bundles from the AS7 distro
- Separate Configuration Admin subsystem

Please also have a look at the latest version of our User Guide.

Enjoy

7.4 JBossOSGi-1.1.1

I am happy to announce the release of JBossOSGi-1.1.1.

You can download the binary here: jboss-osgi-installer-1.1.1.jar

This is a maintenance release for JBoss-AS-7.1.1 integration.

- Add support for target contain AS-7.1.1.Final

Please also have a look at the latest version of our User Guide.

Enjoy

8 Content

8.1 What does MSC stand for?

Modular Service Container. The code lives here

8.2 Why use Blueprint and not Spring directly?

Spring is an implementation and only a single impl exists. OSGi Blueprint is a specification of which multiple implementations exist. Being a specification gives the users the freedom to move from one implementation to another.

8.3 Does JBoss Modules support different versions of a module?

Yes, JBoss Modules has what we call a 'slot'. It is not termed 'version' because not all java projects use the version to associate a compatibility promise with it. You can use the same slot to span multiple major versions. For projects that use an OSGi like version scheme the 'slot' would be equivalent to the 'version'.

8.4 How are the JDK packages like javax.* used by a OSGI bundle?

A bundle must explicitly declare an import on a javax.* package. This allows these packages to be substitutable.

javax.* packages are not by default exported from the JRE, when using javax.* packages there are two options:

- Deploy one or more bundles that export the javax.* packages
- Pick them up from the JRE via the system bundle, to achieve this the org.osgi.framework.system.packages.extra property needs to be set in the AS7 configuration so that these packages are exported via the system bundle.

8.5 Why not use one of the current OSGi Framework implementations (e.g. Felix, Equinox, ...) as the basis for AS7?

JBoss AS7 has been designed around Amdahl's law. This is why it is so blazingly fast. The JBoss OSGi project primarily aims to make OSGi bundles available in the context of AS7, so you can leverage OSGi benefits in connection with Standard JavaEE 6 Technologies. Although JBoss OSGi can be used as a standalone OSGi Runtime, it is not our goal to compete with other runtimes like Felix, Equinox. We use jboss-modules as the integration point with AS7. We hence benefit from the ingenious design and runtime properties. This would not have been possible by using an existing OSGi Framework.

8.6 Why would I opt for JBoss OSGi and not go with Equinox or Felix?

A JavaEE Application Server is the best provider of middleware services. Currently the best one out there is AS7 of course 😉 . A pure OSGi Runtime is a good choice if your project stays within the limits of what such a runtime can provide. It is unreasonable to expect that you can assemble a fully functional, highly integrated, performant "middleware layer" by throwing together a collection of OSGi bundles. A much better strategy in my humble opinion is to opt for an already existing middleware solution with sound OSGi integration. JBoss gives you open choice in that respect.

8.7 Where does project jigsaw fit into all of this ?

Project Jigsaw is being designed to work well with OSGi, but is not a replacement of the mature OSGi technology.

8.8 Is Jigsaw the OSGi core implementation of JBoss AS7

No

8.9 Is there a way to isolate your EJB/Servlet from OSGi APIs, so that dependencies to OSGi services can be injected?

This is currently not supported, but will be in the context of standard Enterprise OSGi JNDI integration. Please monitor JBOSGI-81

8.10 What are the following AS7 milestones, when could we expert OSGi v4.3 support?

We aim for Core 4.3 support by Q3/2012.

8.11 Do you think the integration with standard EE components will be "cleaner" in future versions? Will OSGi be part of specification?

We are working on a standard for integration of OSGi Frameworks with JavaEE Application Servers. There are many hurdles to this and not all of them are technical in nature. We'll publish progress as it occurs.

8.12 How does OSGi module deployment work in a JBoss AS7 domain?

Domain deployments are fully supported. This means you can use the AS 7 management channels to deploy your bundle to multiple servers.

8.13 Where can I find the presentation material from the webinar?

This is documented here

8.14 Are this JBoss OSGi test benchmarks published anywhere?

The benchmark tests can be run as part of the performance testsuite

8.15 Will it be possible to use the same OSGi bundle/services in JBoss AS7 and e.g. Eclipse?

Yes it should be. As long as they don't use any proprietary Eclipse functionality.

8.16 The OSGi API seems clumsy with respect to dependency injection methodologies like CDI. What API should I bind to when I want to use OSGi?

Yes, the OSGi API is not what should be used in user domain code. Instead, opt for Declarative Services, Blueprint. There is also an CDI/OSGi integration that we are working on.

8.17 Folks tell me that productivity drops by orders of magnitude. What are the common pitfalls when working with OSGi?

Design with modularity/dynamicity in mind - be strict about what you provide - be lenient about what you accept. Use OSGi when you need have a modularity issue in large projects. For larger and complex projects you can scale cost linearly. For simple projects you have cost and little benefit.

8.18 Because OSGi is an isolated world with respect to class loading, it is hard to write unit tests for it. What do you recommend as a test strategy?

We use Arquillian, which has a sound OSGi integration. You can learn how to write OSGi tests by looking at our distributed examples.

8.19 What is they key value OSGi modularity provides

Benefits on both technical and business levels.

- Technical
  - encapsulation, no leaking of privates
  - side-by-side deployments: multiple versions of the same library can co-exist
  - memory footprint: libraries are shared across deployments. Only those libraries that are needed are loaded
- Business
  - splitting up your architecture into many modules
  - clear interface improves the scalability of development teams making
  - make it easier to do development efforts in parallel
  - limit the impact of change.

8.20 Is there another feature that's unique to OSGi?

OSGi Services-based applications allow the application to be patched or updated without having to restart the service consumers. This means that if your application is built using OSGi services you can often fix it without taking in down!