JBoss Application Server

Installation And Getting Started Guide

by JBoss Community

Introduction

JBoss Application Server is the open source implementation of the Java EE suite of services. It comprises a set of offerings for enterprise customers who are looking for preconfigured profiles of JBoss Enterprise Middleware components that have been tested and certified together to provide an integrated experience. It's easy-to-use server architecture and high flexibility makes JBoss the ideal choice for users just starting out with J2EE, as well as senior architects looking for a customizable middleware platform.

Because it is Java-based, JBoss Application Server is cross-platform, easy to install and use on any operating system that supports Java. The readily available source code is a powerful learning tool to debug the server and understand it. It also gives you the flexibility to create customized versions for your personal or business use.

Installing JBoss Application Server is simple and easy. You can have it installed and running in no time. This guide will teach you to install and get started with the JBoss Application Server.

1. Help Contribute

If you find a typographical error in the *Installation Guide and Getting Started Guide*, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in JIRA: *http://jira.jboss.com* against the project *JBoss Application Server* and component *Docs/Installation and Getting Started Guide*.

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Note

Be sure to give us your name so you can receive full credit.

Note

This content is taken from svn.jboss.org/repos/jbossas/projects/docs/community/5 and has yet to be branched.

To access the content directly and make changes yourself:

```
  svn co https://svn.jboss.org/repos/jbossas/projects/docs/community/5
--username yourusername
```

What's new In JBossAS 5

1.1. Overview

This is the final release of the JBoss 5.0 series for the Java EE5 codebase that fully complies with the Java EE5 conformance testing certification requirements. It brings us to the end of a 3+ year marathon of redesigning the most popular open-source application server over a completely new kernel architecture, the *JBoss Microcontainer* [http://www.jboss.org/jbossmc]. It also marks the beginning of a new era of innovation for JBoss as we will be exploring the capabilities and limitations of the new architecture in the releases to come. In our view, JBossAS 5 provides a healthy foundation and the most advanced and fully extensible, cross component model, aspect integration, server runtime environment. For information on the APIs that make up Java EE5, see *Java EE APIs* [http://java.sun.com/javaee/5/docs/api/]. A tutorial on Java EE 5 can be found *here* [http://java.sun.com/javaee/5/docs/tutorial/doc/]. Please visit also the *JBoss AS docs* [https://www.jboss.org/community/docs/DOC-12898] pages as we'll be updating the documents with the latest information, and post your questions to the *JBossAS 5 User Forum* [http://www.jboss.com/index.html?module=bb&op=viewforum&f=287].

JBossAS 5 is the next generation of the JBoss Application Server build on top of the new JBoss Microcontainer. The JBoss Microcontainer is a lightweight container for managing POJOs, their deployment, configuration and lifecycle. It is a standalone project that replaces the famous JBoss JMX Microkernel of the 3.x and 4.x JBoss series. The Microcontainer integrates nicely with the JBoss framework for Aspect Oriented Programming, JBoss AOP. Support for JMX in JBoss 5 remains strong and MBean services written against the old Microkernel are expected to work. Further, it lays the groundwork for JavaEE 6 profiles oriented configurations and JBoss AS embedded that will allow for fine grained selection of services for both unit testing and embedded scenarios.

JBossAS 5 is designed around the advanced concept of a Virtual Deployment Framework (VDF), that takes the aspect oriented design of many of the earlier JBoss containers and applies it to the deployment layer. Aspectized Deployers operate in a chain over a Virtual File System (VFS), analyze deployments and produce metadata to be used by the JBoss Microcontainer, which in turn instantiates and wires together the various pieces of a deployment, controlling their lifecycle and dependencies. The VDF allows for both customization of existing component modules including JavaEE and JBoss Microcontainer, as well as introduction of other models such as OSGi and Spring.

1.2. Component Highlights

Many key features of JBoss 5 are provided by integrating other standalone JBoss projects:

- *JBoss Microcontainer* [http://www.jboss.org/jbossmc] is the next generation POJO based kernel that is used as the core of the server. It supports an extensible deployment model and advanced dependency relationships.

- The definition of the non-kernel deployers and deployment is now defined a Profile obtained from the *ProfileService* [http://www.jboss.org/community/docs/DOC-11694]. The ProfileService also provides the ManagementView for *ManagedDeployments/ManagedObjects* [http://www.jboss.org/community/docs/DOC-11349] used by the OpenConsole admin tool.
- JBoss EJB3 included with JBoss 5 provides the implementation of the latest revision of the Enterprise Java Beans (EJB) specification. EJB 3.0 is a deep overhaul and simplification of the EJB specification. EJB 3.0's goals are to simplify development, facilitate a test driven approach, and focus more on writing plain old java objects (POJOs) rather than coding against complex EJB APIs.
- JBoss Messaging is a high performance JMS provider in the JBoss Enterprise Middleware Stack (JEMS), included with JBoss 5 as the default messaging provider. It is also the backbone of the JBoss ESB infrastructure. JBoss Messaging is a complete rewrite of JBossMQ, which is the default JMS provider for the JBoss AS 4.x series.
- JBossCache that comes in two flavors. A traditional tree-structured node-based cache and a PojoCache, an in-memory, transactional, and replicated cache system that allows users to operate on simple POJOs transparently without active user management of either replication or persistency aspects.
- JBossWS is the web services stack for JBoss 5 providing Java EE compatible web services, JAX-WS-2.0.
- JBoss Transactions is the default transaction manager for JBoss 5. JBoss Transactions is founded on industry proven technology and 18 year history as a leader in distributed transactions, and is one of the most interoperable implementations available.
- JBoss Web is the Web container in JBoss 5, an implementation based on Apache Tomcat that includes the Apache Portable Runtime (APR) and Tomcat native technologies to achieve scalability and performance characteristics that match and exceed the Apache Http server.
- JBoss Security has been updated to support pluggable authorization models including SAML, XACML and federation.

JBossAS 5 includes features and bug fixes, many of them carried over upstream from the 4.x codebase. See the *Detailed Release Notes* [https://sourceforge.net/project/shownotes.php?release_id=645033&group_id=22866] section for the full details, and *Section 1.3, “Major Component Upgrades”* for the major component versions included in JBossAS as well as their project page locations.

1.3. Major Component Upgrades

Some rather important JBoss project versions are listed below. You are encouraged to browse the individual project's documentation and view the release notes at www.jboss.org.

- *JBoss Microcontainer* [http://www.jboss.org/jbossmc] v2.0.2.GA
- *JBoss Transactions* [http://www.jboss.org/jbosstm] v4.4.0.GA

- *JBoss WebServices* [http://www.jboss.org/jbossws] v3.0.4.GA
- *JBoss Messaging* [http://www.jboss.org/jbossmessaging] v1.4.1.GA
- *JBoss Web* [http://www.jboss.org/jbossweb] v2.1.1.GA
- *JBoss AOP* [http://www.jboss.org/jbossaop] v2.0.0.SP1
- *JBoss EJB3* [http://www.jboss.org/jbossejb3] v1.0.0-Beta10
- *JBoss Security* [http://www.jboss.org/jbosssecurity] v2.0.2.SP3
- *Hibernate* [http://www.hibernate.org/] v3.3.1.GA
- *Hibernate Entity Manager* [http://www.hibernate.org/] v3.4.0.GA
- *Hibernate Annotations* [http://www.hibernate.org/] v3.4.0.GA
- *JBoss Cache POJO* [http://www.jboss.org/jbosscache] v3.0.0.GA
- *JBoss Cache Core* [http://www.jboss.org/jbosscache] v3.0.1.GA
- *JGroups* [http://www.jboss.org/jgroups] v.2.6.7.GA
- *JGroups* [http://www.jboss.org/jgroups] v.2.6.7.GA
- *JBoss Remoting* [http://www.jboss.org/jbossremoting] v.2.5.0.SP2

For a full list of the JBoss and thirdparty libraries used with JBoss AS 5.0.0.GA check the pom.xml found in the component-matrix directory of the source code distribution. To see the maven dependency tree you can run 'mvn dependency:tree' from the thirdparty directory of the source code distro.

1.4. Project Structure Changes

With the reworking of the server kernel and evolution of various JBoss technologies to indepdnent projects, the JBossAS project is moving towards becoming largely an integration project. Many key pieces are now integrated as thirdparty jars that integration code/configuration makes avaialble as part of a server configuration/profile.

A common theme for JBossAS 5 is the breaking out of internal subsystems into stand-alone projects and the introduction of SPIs throughout the server codebase. Those changes should not affect directly the end user but they are an important part of the JBoss strategy for making available the various EE services as independent projects, so that they can be wired-together and be consumed à la carte inside different runtime environments and not only inside the JBoss Application Server. If you are building JBossAS from source you'll notice we are migrating to a maven2 build. At this point the build is a hybrid one because it declares all JBoss dependencies as maven2 artifacts, however after the dependencies are resolved/imported the legacy ant based build is used to compile and build the distribution. This will change to a full maven build at some point in time. The jboss maven repo can be found *here* [http://repository.jboss.org/maven2/]. Starting from AS5 CR2, please note how the -sources.jar are also downloaded to thirdparty by default. To disable downloading of the sources to thirdparty, define the property skip-download-sources to true either on the command line or in your maven settings.xml.

1.4.1. SVN Information

The project source is rooted at *Anonymous SVN* [http://anonsvn.jboss.org/repos/jbossas/] for public access, and *Committer SVN* [https://svn.jboss.org/repos/jbossas/] for committer access. The directories under these roots follow the usual svn conventions:

- *trunk* [http://anonsvn.jboss.org/repos/jbossas/trunk] - the development branch for the next major version
- *branches* [http://anonsvn.jboss.org/repos/jbossas/branches] - the location for stable branches associated with release series.
- *Branch_5_0* [http://anonsvn.jboss.org/repos/jbossas/branches/Branch_5_0/] - the branch for 5.0.x series development.
- *tags* [http://anonsvn.jboss.org/repos/jbossas/tags] - locations for tagged releases
- *JBoss_5_0_0_GA* [http://anonsvn.jboss.org/repos/jbossas/tags/JBoss_5_0_0_GA/] - The 5.0.0.GA release tag.

1.4.2. The Project Directories

When you checkout the project the resulting subdirectories are:

aspects
: Server aspects

bootstrap
: The server bootstrap that loads the JBoss Microcontainer

build
: The server build directory which contains the main build.xml. See *Section 5.3, “Building with Apache ANT”*Building with ANT for more on building the server.

client
: A maven project that declares the dependcies for the jboss-all-client.jar

cluster
: Clustering related services and integration

component-matrix
: A maven project the declares the external dependencies consumed by the server. This is used to build the thirdparty/* library structure.

connector
: JCA implementation and integration code.

console
: Obsolete admin console. See *JBoss Embedded Console* [http://www.jboss.org/jopr/]project for the future direction of the server admin console.

deployment
: JSR88 deployment services code.

ejb3
: EJB3 integration code.

embedded
: Obsolete JBossAS emebedded project that has been moved to*SVN embedded* [http://anonsvn.jboss.org/repos/jbossas/projects/embedded/] for further development. See the *Design of Embedded JBoss* [http://www.jboss.com/index.html?module=bb&op=viewforum&f=266] forum for design discussions.

hibernate-int
: Hibernate deployment integration code.

iiop
: JacORB integration code for IIOP support.

j2se

jbossas
: JMX remoting and JTS integration code

jmx
: javax.management.* package implementations

jmx-remoting
: A javax.management.remote.JMXConnector implementation

main
: The main() entry point code

management
: JSR77 mbean view generation code

mbeans
: JBoss JMX extensions

messaging
: JBoss Messaging integration code

pom.xml
: The JBossAS root maven pom

profileservice
: The ProfileService, ManagementView, and DeploymentManager implementations.

security	JBoss Security integration code
server	The legacy EJB2 containers, deployers and detached invokers
spring-int	Spring bean deployment integration
system	ProfileServiceBootstrap implementation and management code
system-jmx	MBean service component model and deployers
testsuite	The JBossAS testsuite
thirdparty	The maven2 thirdparty project which builds the local thirdparty jars used by the ant build.
tomcat	JBossWeb integration code and deployers
tools	build tool jars
varia	Various misc services
webservices	JBossWS integration code and deployers

1.5. Configuration Notes

This section describes additional changes in JBossAS 5.

1.5.1. JBoss VFS

JBoss VFS provides a set of different switches to control it's internal behavior. JBoss AS sets boss.vfs.forceCopy=true by default. To see all the provided VFS flags check out the code of the VFSUtils.java class.

- jboss.vfs.forceCopy, useCopyJarHandler option, force copy handling of nested jars if true.
- jboss.vfs.forceVfsJar, true if forcing fallback to vfsjar from default vfszip
- jboss.vfs.forceNoReaper, noReaper option, true if use of the ZipFileLockReaper background closing of ZipFiles should be disabled. VFS uses an internal caching mechanism to speed

up access to deployment artifacts. This means that files in deploy/ remain open as long as they are accessed and then closed by a reaper thread after a 5 seconds inactivity. On window platforms this may cause locking issues if files are re-deployed too quickly. Use jboss.vfs.forceNoReaper=true to disable reaping.

- jboss.vfs.forceCaseSensitive, true if case sensitivity should be enforced
- jboss.vfs.optimizeForMemory, true if zip streams should be kept in memory with their entries in ZipEntry.STORED format.
- jboss.vfs.cache, specifies the org.jboss.util.CachePolicy implementation to use for VFSCache implementations that support an external CachePolicy.

1.5.2. Hibernate Logging

Hibernate-core is now using slf4j-api as a logging facade. To properly integrate that in JBossAS we have created an slf4j-to-jboss-logging adapter (slf4j-jboss-logging.jar) that creates a static binding between sl4j and jboss-logging-spi.

1.5.3. jbossall-client.jar

The client/jbossall-client.jar library that used to bundle the majority of jboss client libraries, is now referencing them instead through the Class-Path manifest entry. This allows swapping included libraries (e.g. jboss-javaee.jar) without having to re-package jbossall-client.jar. On the other hand, it requires that you have jbossall-client.jar together with the other client/*.jar libraries, so they can be found.

1.5.4. EJB3

If using proprietary JBoss/EJB3 annotations, those have moved (since Beta4) into the org.jboss.ejb3.annotation package, EJBTHREE-1099. Those are now included in a new artifact, jboss-ejb3-ext-api.jar

Interoperating with previous JBoss EJB3 implementations may present problems due to serialVersionUIDs issues, EJBTHREE-1118.

Use of JBoss Cache 3.x. has a significantly different API from the 1.x releases used in JBoss AS 4.x and 3.2.x.

@EJB injections should now work from servlets, JBAS-5646.

EJB3 configuration is now controlled by deployers/ejb3.deployer/META-INF/ejb3-deployers-jboss-beans.xml as described in *http://www.jboss.org/community/docs/DOC-12407*

1.5.5. Other JBossAS

The ClassPathExtension MBean has been replaced with a VFS classloader definition, see JBAS-5446.

The old JMX-based ServiceBindingManager has been replaced by a POJO-based ServiceBindingManager, see *AS5ServiceBindingManager Wiki* [http://www.jboss.org/community/docs/DOC-9038].

The Farm service from 4.x has been removed, and replaced with a HASingletonDeploymentScanner that integrates with the ProfileService.

JBoss 5 is stricter when it comes to verifying/deploying JavaEE artifacts. EJB3 deployments that run in AS 4.2 may fail in AS5. We have tried to keep the validation messages as accurate as possible in order to help you modify your deployment descriptors/annotations to be in-line with the JavaEE 5 requirements.

A new jboss.server.log.threshold system property can be used to control the log/server.log threshold. It defaults to DEBUG.

The default conf/jboss-log4j.xml configuration now includes the thread name for entries in log/server.log (JBAS-5274).

The transaction manager configuration has moved from conf/jboss-service.xml to deploy/transaction-service.xml.

All the security related configuration files are now grouped under the deploy/security directory (JBAS-5318). The security configuration changes are further described in *SecurityInJBoss5* [http://www.jboss.org/community/docs/DOC-12199] wiki.

1.5.6. Clustering

A new jboss.jgroups.udp.mcast_port property is to control easy configuration of multicast port. It defaults to ${jboss.jgroups.udp.mcast_port:45688}.

Clustering configurations are now in a deploy/clustering subdirectory

A separate cache is now used for Clustered SSO (JBAS-4676).

Per webapp configuration of useJK, snapshot mode and snapshot interval (JBAS-3460). Default for useJK is whether jvmRoute is set (JBAS-4961).

Total replication (rather than buddy replication) is the default setting for session replication (JBAS-5085).

Loopback is now set to true for all JGroups UDP stacks (JBAS-5323).

1.6. New Configurations

JBossAS 5.0.0.GA introduces two new configuration, the standard and the web config. The standard config is the configuration that has been tested for JavaEE compliance. The major differences with the existing configurations is that call-by-value and deployment isolation are enabled by default, along with support for rmiiiop and juddi (taken from the all config). The configurations that are modified include:

- deployers/ear-deployer-jboss-beans.xml has callByValue and isolated properties set to true
- The "jboss:service=Naming" mbean in conf/jboss-service.xml has CallByValue set to true.
- conf/jndi.properties has java.naming.factory.initial=org.jboss.iiop.naming.ORBInitialContextFactory
- There are additional JacORB configuration files and jars:
  - conf/jacorb.properties
  - deploy/iiop-service.xml
  - lib/avalon-framework.jar
  - lib/jacorb.jar
- A UDDI implementation deployed as deploy/juddi-service.sar

The web config is a new experimental lightweight configuration created around JBoss Web that will follow the developments of the JavaEE 6 web profile. Except for the servlet/jsp container it provides support for JTA/JCA and JPA. It also limits itself to allowing access to the server only through the http port. Please note that this configuration is not JavaEE certified and will most likely change in the following releases.

Another notable change is that the majority of the libraries common to the different configurations have moved to a new shared location, JBOSS_HOME/common/lib/. This is so we avoid having multiple copies of the same libraries in the distribution. The location of the common library directory can be controlled by the following properties:

- jboss.common.base.url defaulting to ${jboss.home.url}/common
- jboss.common.lib.url defaulting to ${jboss.common.base.url}/lib

The common library directory is shared by all the configurations except for the minimal config. It is referenced in the very beginning of every configuration's conf/jboss-service.xml:

```
<classpath codebase="${jboss.server.lib.url}" archives="*"/>
<classpath codebase="${jboss.common.lib.url}" archives="*"/>
```

You can see that the library directory of the individual configurations is still in place, although in some cases it's empty (e.g. JBOSS_HOME/server/default/lib/)

Getting Started

2.1. Pre-Requisites

You must have adequate disk space to install JDK and JBoss Application Server while also allowing enough space for your applications. Before installing JBoss Application Server you must have a working installation of Java. Since JBoss is 100% pure Java you can have it working on any Operating System / Platform that supports Java.

2.1.1. Hardware and Operating System Requirements

For the latest information on supported Operating System / JVM combinations and supported Database platforms, please refer to *http://www.jboss.com*.

2.1.2. Configuring Your Java Environment

You must have a working installation of *JDK 1.5* or *JDK 1.6* before you install JBoss Application Server. You can install the 32-bit or 64-bit JVM as per your requirements. In this guide we will show you how to install a 32-bit Sun JDK 5.0 on a Linux Platform and Microsoft Windows Platform. But before we do that let's take a look at some of the benefits of using a 64-bit JVM.

Benefits of 64-bit JVM on 64-bit OS and Hardware:

- Wider datapath: The pipe between RAM and CPU is doubled, which improves the performance of memory-bound applications.
- 64-bit memory addressing gives virtually unlimited (1 exabyte) heap allocation. However large heaps affect garbage collection.
- Applications that run with more than 1.5GB of RAM (including free space for garbage collection optimization) should utilize the 64-bit JVM.
- Applications that run on a 32-bit JVM and do not require more than minimal heap sizes will gain nothing from a 64-bit JVM. Barring memory issues, 64-bit hardware with the same relative clock speed and architecture is not likely to run Java applications faster than their 32-bit cousin.

Installing and Configuring 32-bit Sun JDK 5.0 or JDK 6.0 on Linux

- Download the Sun JDK 5.0 or JDK 6 (Java 2 Development Kit) from Sun's website: *http://java.sun.com/javase/downloads/index_jdk5.jsp* for JDK 5.0 or *http://java.sun.com/javase/downloads/* for JDK 6.0. Select the JDK Update <x>" (where x is the latest update number) for download and then select "RPM in self-extracting" file for Linux[1]. Read the instructions on Sun's website for installing the JDK.
- If you do not want to use SysV service scripts you can install the "self-extracting file" for Linux instead of choosing the "RPM in self-extracting" file. In that case you can skip the next step mentioned here. But it is recommended to use the SysV service scripts for production servers.

- Download and install the appropriate `-compat` `RPM` from `JPackage` *here* [ftp://jpackage.hmdc.harvard.edu/JPackage/1.7/generic/RPMS.non-free/]. Please ensure you choose a matching version of the `-compat` package to the JDK you installed.
- Create an environment variable that points to the JDK installation directory and call it `JAVA_HOME`. Add `$JAVA_HOME/bin` to the system path to be able to run `java` from the command line. You can do this by adding the following lines to the `.bashrc` file in your home directory.

```
#In this example /usr/java/jdk1.6.0_07 is the JDK installation directory.
                        export JAVA_HOME=/usr/java/jdk1.6.0_07
                        export PATH=$PATH:$JAVA_HOME/bin
```

  Set this variable for the user account doing the installation and also for the user account that will run the server.

- If you have more than one version of JVM installed in your machine, make sure you are using the JDK1.5 or JDK1.6 installation as the default source for the `java` and `javac` executables. You can do this using the alternatives system. The alternatives system allows different versions of Java, from different sources to co-exist on your system.

Select alternatives for java, javac and java_sdk_1.<x>

- As root, type the following command at the shell prompt and you should see something like this:

```

[root@vsr ~]$ /usr/sbin/alternatives --config java
There are 2 programs which provide 'java'.
Selection    Command
-----------------------------------------------
1          /usr/lib/jvm/jre-1.4.2-gcj/bin/java
*+ 2          /usr/lib/jvm/jre-1.5.0-sun/bin/java
Enter to keep the current selection[+], or type selection number:
```

  Make sure the Sun version [`jre-1.5.0-sun` in this case] is selected (marked with a '+' in the output), or select it by entering its number as prompted.

- Repeat the same for javac and java_sdk_1.<x>

```

[root@vsr ~]$ /usr/sbin/alternatives --config javac
There are 1 programs which provide 'javac'.
```

```
Selection   Command
-----------------------------------------------
*+ 1           /usr/lib/jvm/java-1.5.0-sun/bin/javac
Enter to keep the current selection[+], or type selection number:
```

```
[root@vsr ~]$ /usr/sbin/alternatives --config java_sdk_1.5.0
There are 1 programs which provide 'java_sdk_1.5.0'.
Selection   Command
-----------------------------------------------
*+ 1           /usr/lib/jvm/java-1.5.0-sun
Enter to keep the current selection[+], or type selection number:
```

You should verify that java, javac and java_sdk_1.<x> all point to the same manufacturer and version.

Note

You can always override this step by setting the `JAVA_HOME` environment variable as explained in the previous step.

- Make sure that the `java` executable is in your path and that you are using an appropriate version. To verify your Java environment, type `java -version` at the shell prompt and you should see something like this:

```
[root@vsr ~]$ java -version
java version "1.5.0_14"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_14-b03)
Java HotSpot(TM) Client VM (build 1.5.0_14-b03, mixed mode, sharing)
```

Installing and Configuring 32-bit Sun JDK 5.0 or JDK 6.0 on Microsoft Windows

- Download the Sun JDK 5.0 (Java 2 Development Kit) from Sun's website: *http://java.sun.com/javase/downloads/index_jdk5.jsp* for JDK 5.0 or *http://java.sun.com/javase/downloads/* for JDK 6.0. Choose the JDK Update <x>" (where x is the latest update number) for download and then select your Windows Platform options to perform the installation.

- Create an environment variable called `JAVA_HOME` that points to the JDK installation directory, for example: `C:\Program  Files\Java\jdk1.5.0_14\`. In order to run java from the command line add the `jre\bin` directory to your path, for example: `C:\Program Files\Java\jdk1.5.0_14\jre\bin`. To do this, open the Control Panel from the Start Menu, switch to Classic View if necessary, open the System Control Panel applet, select the Advanced Tab, and click on the Environment Variables button.

Installation Alternatives

You can install the JBoss Application Server in one of these two modes:

- *Binary files download*

  In this form of installation, simply unzip the downloaded zip file to the directory of your choice. You can unzip the JBoss Application Server on any operating system that supports the zip format. The zip file is available on *http://labs.jboss.com/jbossas/downloads/*. Please ensure you have met the pre-requisites required before proceeding with your installation. Pre-requisites are discussed in *Section 2.1, “Pre-Requisites”*. Further details on installation using the Binary files are discussed in *Chapter 4, Installation With Binary Download*

  JBossAS 5.0.0 can be compiled with both Java5 and Java6. The Java5 compiled binary is our primary/recommended binary distribution. It has undergone rigorous testing and can run under both a Java 5 and a Java 6 runtime. When running under Java 6 you need to manually copy the following libraries from the `JBOSS_HOME/client` directory to the `JBOSS_HOME/lib/endorsed` directory, so that the JAX-WS 2.0 apis supported by JBossWS are used:

  ```
  * jbossws-native-saaj.jar
                      * jbossws-native-jaxrpc.jar
                      * jbossws-native-jaxws.jar
                      * jbossws-native-jaxws-ext.jar
  ```

  Another alternative is to download the jdk6 distribution *(`jboss-5.0.0.CR2-jdk6.zip`)* [http://downloads.sourceforge.net/jboss/jboss-5.0.0.CR2-jdk6.zip?modtime=1221686600&big_mirror=1] in which case no configuration changes are required. Please refer to the *release notes* [http://sourceforge.net/project/shownotes.php?release_id=627020&group_id=22866] for additional information about running with JDK 6.

- *Source Files download*

  In this form of installation, download the source files from the web and build the source files locally. On successfully building your source files you can manually copy the built file into a desired folder and start the server. Please ensure you have met the pre-requisites required before proceeding with your installation. Pre-requisites are discussed in *Section 2.1, “Pre-Requisites”*. For more instructions on building your source files, please refer to *Chapter 5, Installation With Source Download*.

Three types of server configurations will be included in your installation - *minimal*, *default*, and *all*.

Installation With Binary Download

4.1. Download and Installation

You can download the Binary zip files from *http://labs.jboss.com/jbossas/downloads/*.

There are two binary distributions available:

1. *jboss-5.0.0.CR2.zip* [http://downloads.sourceforge.net/jboss/jboss-5.0.0.CR2.zip?modtime=1221686752&big_mirror=1]

2. *jboss-5.0.0.CR2-jdk6.zip* [http://downloads.sourceforge.net/jboss/jboss-5.0.0.CR2-jdk6.zip?modtime=1221686600&big_mirror=1]

In this form of installation, simply unzip the downloaded zip file to the directory of your choice on any operating system that supports the zip format.

- Unzip `jboss-<release>.zip` to extract the archive contents into the location of your choice. You can do this using the JDK `jar` tool (or any other ZIP extraction tool). In the example below we are assuming you downloaded the zip file to the `/jboss` directory.

```
[usr]$ cd /jboss
[usr]$ jar -xvf jboss-<release>.zip
```

- You should now have a directory called `jboss-<release>`. Next you need to set your JBOSS_HOME environment variables. This is discussed in *Chapter 6, Setting the JBOSS_HOME variable*.

Installation With Source Download

5.1. Download and Installation

You can download the zip source file from *http://labs.jboss.com/jbossas/downloads/*.

- Uncompress `jboss-<release>-src.tar.gz` to extract the archive contents into the location of your choice. You can do this using the `tar` archiving utility in Linux (or any other compatible extraction tool). In this example we are assuming your source files were copied in the `/jboss` folder.

  ```
  [user@localhost]$ cd /jboss
                              [user@localhost]$ tar -xvf
   jboss-<release>-src.tar.gz
  ```

- You should now have a directory called `jboss-<release>-src.tar.gz`. The next step is to build your source files. In this example we are using Apache ANT. This is discussed in the following section.

5.2. Installing and configuring ANT

Apache Ant [http://ant.apache.org/] is a Java-based build tool. Instead of using an extended model using shell-based commands, Ant is extended using Java classes that use XML-based configuration files. The configuration files call out a target tree that executes various tasks. Each task is run by an object that implements a particular Task interface. This gives you the ability to perform cross platform builds. Please also note that if needed, Ant provides an <exec> task that allows commands to be executed based on the Operating System it is executing on. For more information on Apache ANT please click *here* [http://ant.apache.org/].

You will need to build your JBoss Application Server source files before you can run the application server. Apache Ant is shipped with the JBoss Application Server source files and can be executed from the `<source_directory>/tools/bin` directory.

The source files can also be built using Apache Maven which is also shipped with the JBoss Application Server source files under `<source_directory>/tools/maven` directory. For more information about Apache Maven, please refer to *http://maven.apache.org/* [].

Like Java, you also need to set the environment variables for Apache ANT and/or Apache Maven. The following example illustrates a desirable configuration for the `.bashrc` file. In the example the file is edited using the gnome text editor (`gedit`).

```
[user@localhost ~]$ gedit .bashrc

                # Source global definitions
                if [ -f /etc/bashrc ]; then
```

```
               /etc/bashrc
               fi
               ......
               # User specific aliases and functions
               # The following are the environment variables for Java , ANT
and Maven

               export JAVA_HOME=/usr/java/jdk1.6.0_07/
               export PATH=$PATH:$JAVA_HOME/bin

               export
ANT_HOME=/home/downloads/jboss-<source_directory>/tools/
               export PATH=$PATH:$ANT_HOME/bin

               export
MAVEN_HOME=/home/downloads/jboss-<source_directory>/tools/maven
               export PATH=$PATH:$MAVEN_HOME/bin
```

To implement the changes you've made to the `.bashrc` file, type the following on a terminal.

```
[user@localhost ~]$ source .bashrc
                [user@localhost ~]$
```

If any errors are displayed, please check your `.bashrc` file for errors and ensure that all directory paths are correct.

5.3. Building with Apache ANT

To build the JBoss Application Server source files with Apache ANT, from a terminal change directory to where the unzipped source files are. In the following example we are assuming that the source files were copied and unzipped in the logged in user's `downloads` folder.

```
[user@localhost]$ cd /home/user/downloads/jboss-<release>-src/build
[504][valkyrie: jboss-5.0.0.GA-src]$ ls
aspects            hibernate-int   security
bootstrap          iiop            server
build     j2se           spring-int
client     jbossas        system
cluster     jmx            system-jmx
component-matrix  jmx-remoting    testsuite
connector          main            thirdparty
console            management      tomcat
deployment         mbeans          tools
docbook-support    messaging       varia
ejb3               pom.xml         webservices
embedded           profileservice
```

From the contents of the `build` directory above, you can see the `build.xml` file which is used by Apache ANT as a configuration file when building your source files. The next step is to perform the build using Apache ANT as illustrated below.

```
[571][valkyrie: build]$ ant
Buildfile: build.xml

_buildmagic:init:
Trying to override old definition of task property

_buildmagic:init:local-properties:
[copy] Copying 1 file to /Users/svn/Releases/jboss-5.0.0.GA-src/build

_buildmagic:init:buildlog:

configure:
[echo] groups:  default
[echo] modules:
 bootstrap,main,j2se,mbeans,jmx,system,system-
jmx,security,server,deployment,jbossas/remoting,jmx-remoting,jbossas/jmx-
remoting,messaging,cluster,varia,iiop,aspects,profileservice,connector,management,ejb3,tomca
int,console,spring-int

...

createthirdparty:
[echo] Calling mvn command located in
 /Users/svn/Releases/jboss-5.0.0.GA-src/build/../tools/maven

...
main:

BUILD SUCCESSFUL
Total time: 21 minutes 34 seconds
```

A successful build will have the above message. The first time you build the tree it will download a large number of thirdparty files from maven repositories. After that, these will be used from the local repository and the build will be much faster. Typical initial build times can be 30 minutes with subsequent build times 3 minutes. If your build fails, please check the error log and ensure that your configuration files and environment variables are correctly set. The JBoss Application Server files are built under the `build/output/jboss-<release>` directory as indicated below.

Note

At this point the JBoss Application Server source files build is a hybrid one (builds in both Ant and Maven) because it declares all JBoss dependencies as maven2 artifacts, however after the dependencies are resolved/imported the legacy ant based build is used to compile and build the distribution. The JBoss Application Server source files will change to a full maven build soon.

```
[578][valkyrie: build]$ ls
VersionRelease.java build.sh  local.properties
build-distr.xml  build.xml  output
build-release.xml docs   pom.xml
build.bat  eclipse.psf
build.log  etc
[579][valkyrie: build]$ ls output/
jboss-5.0.0.GA
[580][valkyrie: build]$
```

The `jboss-<release>` directory contains your successful JBoss Application Server files. You can copy this folder to a different location or run the server from this folder after setting the JBOSS_HOME environment variable in your `.bashrc` file. Next you need to set your JBOSS_HOME environment variables. This is discussed in *Chapter 6, Setting the JBOSS_HOME variable*.

5.4. Java6 Notes

JBossAS 5.0.0.GA can be compiled with both Java5 and Java6. The Java5 compiled binary is our primary/recommended binary distribution. It has undergone rigorous testing and can run under both a Java 5 and a Java 6 runtime. When running under Java 6 you need to manually copy the following libraries from the JBOSS_HOME/client directory to the JBOSS_HOME/lib/endorsed directory, so that the JAX-WS 2.0 apis supported by JBossWS are used:

jbossws-native-saaj.jar
jbossws-native-jaxrpc.jar
jbossws-native-jaxws.jar
jbossws-native-jaxws-ext.jar
The other option is to download the jdk6 distribution (jboss-5.0.0.GA-jdk6.zip) in which case no configuration changes are required. If you still have problems using JBoss with a Sun Java 6 runtime, you may want to set -Dsun.lang.ClassLoader.allowArraySyntax=true, as described in JBAS-4491. Other potential problems under a Java 6 runtime include:

ORB getting prematurely destroyed when using Sun JDK 6 (see Sun Bug ID: 6520484)
Unimplemented methods in Hibernate for JDK6 interfaces.
When JBossAS 5 is compiled with Java 6, support for the extended JDBC 4 API is included in the binary, however this can only be used under a Java 6 runtime. In this case no manual configuration steps are necessary.

Note

It should be noted that the Java 6 compiled distribution of JBoss AS 5 is still in an experimental stage in terms of testing.

Setting the JBOSS_HOME variable

6.1. Setting the `JBOSS_HOME` variable in Linux.

Before you can run the JBoss Application Server, you need to ensure that you've configured the JBOSS_HOME environment variable in your `.bashrc` file as follows. In this example the Application Server folder has beeen copied to the `/usr/jboss/jboss-<release>` folder. The following is a `.bashrc` file used in this installation. Please ensure that your `.bashrc` file has a similar configuration.

```
[user@localhost ~]$ gedit .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
/etc/bashrc
fi
......
# User specific aliases and functions
# The following are the environment variables for Java, ANT and JBoss

export JAVA_HOME=/usr/java/jdk1.6.0_07
export PATH=$PATH:$JAVA_HOME/bin

export ANT_HOME=/usr/ant/apache-ant-1.6.0
export PATH=$PATH:$ANT_HOME/bin

export JBOSS_HOME=/usr/jboss/jboss-<release>
export PATH=$PATH:$JBOSS_HOME/bin
```

To implement your `.bashrc` file changes run the following command.

```
[user@localhost ~]$ source .bashrc
[user@localhost ~]$
```

If no errors are displayed on your terminal, you are now ready to run your JBoss Application Server.

6.2. Setting the `JBOSS_HOME` variable in Windows.

- Create an environment variable called `JBOSS_HOME` that points to the JBoss Application Server installation directory, for example: `C:\Program Files\JBoss\jboss-<release>\`.
- In order to run JBoss Application Server from the command line, add the `jboss-<release>\bin` directory to your path, for example: `C:\Program Files\JBoss\jboss-<release>\bin`. To do this, open the Control Panel from the Start Menu, switch to Classic View if necessary, open the System Control Panel applet, select the Advanced Tab, and click on the Environment Variables button.

You are now ready to start the JBoss Application Server.

Uninstall JBoss

The JBoss Application Server may be uninstalled by simply deleting the JBoss Application Server's installation directory. You will also need to remove the *JBOSS_HOME* environment variables discussed in *Chapter 6, Setting the JBOSS_HOME variable* for your Linux or Windows platform.

Test your Installation

After you have installed the JBoss Application Server, it is wise to perform a simple startup test to validate that there are no major problems with your Java VM/operating system combination. To test your installation, open the `JBOSS_DIST/jboss-<release>/bin` directory and execute the `run.bat` (for Windows) or `run.sh` (for Linux) script, as appropriate for your operating system.

Your output should look similar to the following (accounting for installation directory differences) and contain no error or exception messages:

```
       [samson@dhcp-1-150 bin]$ sh run.sh

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

       JBoss Bootstrap Environment

       JBOSS_HOME:
/home/svn/JBossHead/jboss-head/build/output/jboss-5.0.0.GA

       JAVA: /Library/Java/Home/bin/java

       JAVA_OPTS: -Dprogram.name=run.sh -Xms128m -Xmx512m
-XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true
-Dsun.rmi.dgc.client.gcInterval=3600000
-Dsun.rmi.dgc.server.gcInterval=3600000

       CLASSPATH:
/home/svn/JBossHead/jboss-head/build/output/jboss-5.0.0.GA/bin/run.jar


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

       21:47:23,874 INFO  [ServerImpl] Starting JBoss (Microcontainer)...
       21:47:23,875 INFO  [ServerImpl] Release ID: JBoss [Morpheus]
5.0.0.GA (build: SVNTag=JBoss_5_0_0_GA date=200812011226)
       .
       .
       ...output truncated
       .
       .
       21:47:46,090 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on
ajp-127.0.0.1-8009
       21:47:46,112 INFO  [ServerImpl] JBoss (Microcontainer) [5.0.0.GA
(build: SVNTag=JBoss_5_0_0_GA date=200812011226)] Started in 22s:227ms
```

Now open `http://localhost:8080` in your web browser. (Make sure you dont have anything else already on your machine using that port).[1] The contents of your page should look similar to the following: *Figure 8.1, “Test your Installation”*.

Figure 8.1. Test your Installation

You are now ready to use the JBoss Application Server.

[1] Note that on some machines, the name localhost may not resolve properly and you may need to use the local loopback address 127.0.0.1 instead.

The JBoss Server - A Quick Tour

9.1. Server Structure

Now that you've downloaded JBoss and have run the server for the first time, the next thing you will want to know is how the installation is laid out and what goes where. At first glance there seems to be a lot of stuff in there, and it's not obvious what you need to look at and what you can safely ignore for the time being. To remedy that, we'll explore the server directory structure, locations of the key configuration files, log files, deployment and so on. It's worth familiarizing yourself with the layout at this stage as it will help you understand the JBoss service architecture so that you'll be able to find your way around when it comes to deploying your own applications.

9.2. Server Configurations

Fundamentally, the JBoss architecture consists of the microcontainer, bootstrap beans loaded into the micrcontainer, a collection of deployers for loading various deployment types, and various mcbean(-jboss-beans.xml) and legacy mbean(jboss-service.xml) deployments. This makes it easy to assemble different configurations and gives you the flexibility to tailor them to meet your requirements.

You don't have to run a large, monolithic server all the time; you can remove the components you don't need (which can also reduce the server startup time considerably) and you can also integrate additional services into JBoss by writing your own MBeans. You certainly do not need to do this to be able to run standard Java EE 5 applications though.

You don't need a detailed understanding of the microcontainer to use JBoss, but it's worth keeping a picture of this basic architecture in mind as it is central to the way JBoss works.

The JBoss Application Server ships with three different server configurations. Within the `<JBoss_Home>/server` directory, you will find five subdirectories: `minimal`, `default`, `standard`, `all` and `web` - one for each server configuration. Each of these configurations provide a different set of services. The `default` configuration is the one used if you don't specify another one when starting up the server.

minimal
: has a minimal configuration—the bare minimum services required to start JBoss. It starts the logging service, a JNDI server and a URL deployment scanner to find new deployments. This is what you would use if you want to use JMX/JBoss to start your own services without any other Java EE 5 technologies. This is just the bare server. There is no web container, no EJB or JMS support.

default
: is a base Java EE 5 server profile containing a default set of services. It has the most frequently used services required to deploy a Java EE application. It does not include the JAXR service, the IIOP service, or any of the clustering services.

all
: The all configuration starts all the available services. This includes the RMI/IIOP and clustering services, which are not loaded in the default configuration.

standard
: is the JavaEE 5 certified configuration of services.

web
: is a lightweight web container oriented profile that previews the JavaEE 6 web profile.

If you want to know which services are configured in each of these instances, the primary differences will be in the `<JBoss_Home>/server/<instance-name>/deployers/` directory and also the services deployments in the `<JBoss_Home>/server/<instance-name>/deploy` directory. For example, the default profile deployers and deploy directory contents are:

```
[usr@localhost <JBoss_Home>]$ls server/default/deployers
alias-deployers-jboss-beans.xml  jboss-aop-jboss5.deployer
bsh.deployer    jboss-jca.deployer
clustering-deployer-jboss-beans.xml  jbossweb.deployer
dependency-deployers-jboss-beans.xml jbossws.deployer
directory-deployer-jboss-beans.xml j sr77-deployers-jboss-beans.xml
ear-deployer-jboss-beans.xml  metadata-deployer-jboss-beans.xml
ejb-deployer-jboss-beans.xml  seam.deployer
ejb3.deployer    security-deployer-jboss-beans.xml
hibernate-deployer-jboss-beans.xml
[usr@localhost <JBoss_Home>]$ls server/default/deploy
ROOT.war    jsr88-service.xml
cache-invalidation-service.xml legacy-invokers-service.xml
ejb2-container-jboss-beans.xml mail-ra.rar
ejb2-timer-service.xml  mail-service.xml
ejb3-connectors-jboss-beans.xml management
ejb3-container-jboss-beans.xml messaging
ejb3-interceptors-aop.xml  monitoring-service.xml
ejb3-timer-service.xml  profileservice-jboss-beans.xml
hdscanner-jboss-beans.xml  properties-service.xml
hsqldb-ds.xml   quartz-ra.rar
http-invoker.sar   remoting-jboss-beans.xml
jboss-local-jdbc.rar  schedule-manager-service.xml
jboss-xa-jdbc.rar   scheduler-service.xml
jbossweb.sar   security
jbossws.sar   sqlexception-service.xml
jca-jboss-beans.xml  transaction-jboss-beans.xml
jms-ra.rar   transaction-service.xml
jmx-console.war   uuid-key-generator.sar
```

```
jmx-invoker-service.xml  vfs-jboss-beans.xml
jmx-remoting.sar
```

while the web profile deployers and deploy directory contents are:

```

[usr@localhost <JBoss_Home>]$ls server/web/deployers
alias-deployers-jboss-beans.xml jbossweb.deployer
ejb3.deployer   metadata-deployer-jboss-beans.xml
jboss-aop-jboss5.deployer  security-deployer-jboss-beans.xml
jboss-jca.deployer
[usr@localhost <JBoss_Home>]$ls server/web/deployers
ROOT.war    jbossweb.sar
ejb3-container-jboss-beans.xml jca-jboss-beans.xml
hdscanner-jboss-beans.xml  jmx-console.war
hsqldb-ds.xml   jmx-invoker-service.xml
http-invoker.sar   security
jboss-local-jdbc.rar  transaction-jboss-beans.xml
jboss-xa-jdbc.rar
```

Note

The **default** configuration is the one used if you don't specify another one when starting up the server.

To start the server using an alternate configuration refer to *Section 9.3.2, "Start the Server With Alternate Configuration"*.

9.2.1. Server Configuration Directory Structure

The directory server configuration you're using, is effectively the server root while JBoss is running. It contains all the code and configuration information for the services provided by the particular server configuration. It's where the log output goes, and it's where you deploy your applications. *Table 9.1, "Server Configuration Directory Structure"* shows the directories inside the server configuration directory (`<JBoss_Home>/server/<instance-name>`) and their functions.

Table 9.1. Server Configuration Directory Structure

Directory	Description
`conf`	The `conf` directory contains the `bootstrap.xml` bootstrap descriptor file for a given server configuration. This defines the core microcontainer beans that are fixed for the lifetime of the server.

Directory	Description
`data`	The `data` directory is available for use by services that want to store content in the file system. It holds persistent data for services intended to survive a server restart. Serveral JBoss services, such as the embedded Hypersonic database instance, store data here.
`deploy`	The `deploy` directory contains the hot-deployable services (those which can be added to or removed from the running server). It also contains applications for the current server configuration. You deploy your application code by placing application packages (JAR, WAR and EAR files) in the `deploy` directory. The directory is constantly scanned for updates, and any modified components will be re-deployed automatically.
`lib`	This directory contains JAR files (Java libraries that should not be hot deployed) needed by this server configuration. You can add required library files here for JDBC drivers etc. All JARs in this directory are loaded into the shared classpath at startup. Note that this directory only contains those jars unique to the server configuration. Jars common across the server configurations are now located in `<JBoss_Home>/common/lib`.
`log`	This is where the log files are written. JBoss uses the Jakarta `log4j` package for logging and you can also use it directly in your own applications from within the server. This may be overridden through the `conf/jboss-log4j.xml` configuration file.
`tmp`	The `tmp` directory is used for temporary storage by JBoss services. The deployer, for example, expands application archives in this directory.
`work`	This directory is used by Tomcat for compilation of JSPs.

9.2.2. The "default" Server Configuration File Set

The "`default`" server configuration file set is located in the `<JBoss_Home>/server/default` directory. The following example illustrates a truncated directory structure of the `jboss-as-<release>` server configuration files:

```
[user@localhost <JBoss_Home>]$ tree
|-- bin
|-- client
|-- common
|   |-- lib
|   |   |-- antlr.jar
|   |   |-- ... many more jars
|-- docs
|   |-- dtd
|   |-- examples
|   |   |-- binding-manager
|   |   |   `-- sample-bindings.xml
|   |   |-- jca
|   |   |-- jms
|   |   |-- jmx
|   |   |-- netboot
|   |   |   `-- netboot.war
|   |   `-- varia
|   |       |-- deployment-service
|   |       |-- derby-plugin.jar
|   |       |-- entity-resolver-manager
|   |       |   `-- xmlresolver-service.xml
|   |       `-- jboss-bindings.xml
|   `-- schema
|-- lib
|   |-- commons-codec.jar
|   |-- commons-httpclient.jar
|   |-- commons-logging.jar
|   |-- concurrent.jar
|   |-- endorsed
|   |   |-- serializer.jar
|   |   |-- xalan.jar
|   |   `-- xercesImpl.jar
|   |-- getopt.jar
|   |-- jboss-common.jar
|   |-- jboss-jmx.jar
|   |-- jboss-system.jar
|   |-- jboss-xml-binding.jar
|   `-- log4j-boot.jar
`-- server
|-- all
|   |-- conf
|   |   |-- bootstrap/
|   |   |   |-- aop.xml
|   |   |   |-- bindings.xml
|   |   |   |-- aop.xml
|   |   |   |-- classloader.xml
|   |   |   |-- deployers.xml
|   |   |   |-- jmx.xml
|   |   |   |-- profile-repository.xml
|   |   |   |-- profile.xml
|   |   |   |-- vfs.xml
|   |   |-- bootstrap.xml
|   |   |-- bootstrap-norepo.xml
|   |   |-- jacorb.properties
|   |   |-- java.policy
|   |   |-- jax-ws-catalog.xml
|   |   |-- jboss-log4j.xml
|   |   |-- jboss-service.xml
|   |   |-- jbossjta-properties.xml
|   |   |-- jndi.properties
|   |   |-- login-config.xml
|    |    |-- props
|    |    |    |-- jbossws-roles.properties
|    |    |    |-- jbossws-users.properties
|    |    |    |-- jmx-console-roles.properties
|    |    |    `-- jmx-console-users.properties
|    |    |-- standardjboss.xml
|    |    |-- standardjbosscmp-jdbc.xml
|    |    `-- xmdesc
|    |-- deploy
|    |-- deploy-hasingleton
|    |    `-- jms
|    |-- deployers
|    `-- lib
|-- default
|    |-- conf
|    |    |-- bootstrap/
|    |    |    |-- aop.xml
|    |    |    |-- bindings.xml
|    |    |    |-- aop.xml
|    |    |    |-- classloader.xml
|    |    |    |-- deployers.xml
|    |    |    |-- jmx.xml
|    |    |    |-- profile-repository.xml
|    |    |    |-- profile.xml
|    |    |    |-- vfs.xml
|    |    |-- bootstrap.xml
|    |    |-- bootstrap-norepo.xml
|    |    |-- jacorb.properties
|    |    |-- java.policy
|    |    |-- jax-ws-catalog.xml
|    |    |-- jboss-log4j.xml
|    |    |-- jboss-service.xml
|    |    |-- jbossjta-properties.xml
|    |    |-- jndi.properties
|    |    |-- login-config.xml
|    |    |-- props
|    |    |    |-- jbossws-roles.properties
|    |    |    |-- jbossws-users.properties
|    |    |    |-- jmx-console-roles.properties
|    |    |    `-- jmx-console-users.properties
|    |    |-- standardjboss.xml
|    |    |-- standardjbosscmp-jdbc.xml
|    |    `-- xmdesc
|    |         |-- AttributePersistenceService-xmbean.xml
|    |         |-- ClientUserTransaction-xmbean.xml
|    |         |-- JNDIView-xmbean.xml
|    |         |-- Log4jService-xmbean.xml
|    |         |-- NamingBean-xmbean.xml
|    |         |-- NamingService-xmbean.xml
|   |       |-- TransactionManagerService-xmbean.xml
|   |       |-- org.jboss.deployment.JARDeployer-xmbean.xml
|   |       |-- org.jboss.deployment.MainDeployer-xmbean.xml
|   |       `-- org.jboss.deployment.SARDeployer-xmbean.xml
|   |-- data
|   |   |-- hypersonic
|   |   |-- jboss.identity
|   |   |-- tx-object-store
|   |   `-- xmbean-attrs
|   |-- deploy
|   |-- lib
|   |-- log
|   |   |-- boot.log
|   |   |-- server.log
|   |   `-- server.log.2008-08-09
|   |-- tmp
|   `-- work
|       `-- jboss.web
|           `-- localhost
`-- minimal
|-- conf
|   |-- bootstrap/
|   |-- bootstrap/aop.xml
|   |-- bootstrap/classloader.xml
|   |-- bootstrap/deployers.xml
|   |-- bootstrap/jmx.xml
|   |-- bootstrap/profile.xml
|   |-- bootstrap.xml
|   |-- jboss-log4j.xml
|   |-- jboss-service.xml
|   |-- jndi.properties
|   `-- xmdesc
|       |-- NamingBean-xmbean.xml
|       `-- NamingService-xmbean.xml
|-- deploy/
|-- deploy/hdscanner-jboss-beans.xml
|-- deployers/
`-- lib
|-- jboss-minimal.jar
|-- jnpserver.jar
`-- log4j.jar
```

9.2.2.1. Contents of "conf" directory

The files in the `conf` directory are explained in the following table.

Table 9.2. Contents of "conf" directory

File	Description
`bootstrap.xml`	This is the `bootstrap.xml` file that defines which additional microcontainer deployments will be loaded as part of the bootstrap phase.
`bootstrap/*`	This directory contains the microcontainer bootstrap descriptors that are referenced from the `bootstrap.xml` file.
`jboss-service.xml`	`jboss-service.xml` legacy core mbeans that have yet to be ported to either bootstrap deployments, or deploy services. This file will go away in the near future.
`jbossjta-properties.xml`	`jbossjta-properties.xml` specifies the JBossTS transaction manager default properties.
`jndi.properties`	The `jndi.properties` file specifies the JNDI `InitialContext` properties that are used within the JBoss server when an `InitialContext` is created using the no-arg constructor.
`java.policy`	A placeholder java security policy file that simply grants all permissions.
`jboss-log4j.xml`	This file configures the Apache log4j framework category priorities and appenders used by the JBoss server code.
`login-config.xml`	This file contains sample server side authentication configurations that are applicable when using JAAS based security.
`props/*`	The `props` directory contains the users and roles property files for the `jmx-console`.
`standardjboss.xml`	This file provides the default container configurations.
`standardjbosscmp-jdbc.xml`	This file provides a default configuration file for the JBoss CMP engine.
`xmdesc/*-mbean.xml`	The `xmdesc` directory contains XMBean descriptors for several services configured in the `jboss-service.xml` file.

9.2.2.2. Contents of "deployers" directory

The files in the `deployers` directory are explained in the following table.

Table 9.3. Contents of "deployers" directory

File	Description
`alias-deployers-jboss-beans.xml`	Deployers that know how to handle The know how to handle <alias> in <deployment> as true controller context. Meaning they will only get active/installed when their original is installed.
`bsh.deployer`	This file configures the bean shell deployer, which deploys bean shell scripts as JBoss mbean services.
`clustering-deployer-jboss-beans.xml`	Clustering-related deployers which add dependencies on needed clustering services to clustered EJB3, EJB2 beans and to distributable web applications.
`dependency-deployers-jboss-beans.xml`	Deployers for aliases.txt, jboss-dependency.xml jboss-depedency.xml adds generic dependency on whatever. aliases.txt adds human-readable name for deployments, e.g. vfszip://home/blah/.../jboss-5.0.0.GA/ server/default/deploy/some-long-name.ear aliased to ales-app.ear.
`directory-deployer-jboss-beans.xml`	Adds legacy behavior for directories, handling its children as possible deployments. e.g. .sar's lib directory to treat its .jar files as deployments
`ear-deployer-jboss-beans.xml`	JavaEE 5 enterprise application related deployers
`ejb-deployer-jboss-beans.xml`	Legacy JavaEE 1.4 ejb jar related deployers
`ejb3.deployer`	This is a deployer that supports JavaEE 5 ejb3, JPA, and application client deployments, .
`hibernate-deployer-jboss-beans.xml`	Deployers for Hibernate -hibernate.xml descriptors, which are similar to Hibernate's .cfg.xml files.
`jboss-aop-jboss5.deployer`	JBossAspectLibrary and base aspects. Why is this in deployers, dependencies?
`jboss-jca.deployer`	`jboss-jca.deployer` description
`jbossweb.deployer`	The JavaEE 5 servlet, JSF, JSP deployers.
`jbossws.deployer`	The JavaEE 5 webservices endpoint deployers.
`jsr77-deployers-jboss-beans.xml`	Deployers for creating the JSR77 MBeans from the JavaEE components.

File	Description
`metadata-deployer-jboss-beans.xml`	Deployers for processing the JavaEE metadata from xml, annotations.
`seam.deployer`	Deployer providing integration support for JBoss Seam applications.
`security-deployer-jboss-beans.xml`	Deployers for configuration the security layers of the JavaEE components.

9.2.2.3. Contents of "deploy" directory

The files in the `deploy` directory are explained in the following table.

Table 9.4. Contents of "deploy" directory

File	Description
`ROOT.war`	`ROOT.war` establishes the '/' root web application.
`cache-invalidation-service.xml`	This is a service that allows for custom invalidation of the EJB caches via JMS notifications. It is disabled by default.
`ejb2-container-jboss-beans.xml`	`ejb2-container-jboss-beans.xml` UserTransaction integration bean for the EJB2 containers.
`ejb2-timer-service.xml`	`ejb2-timer-service.xml` contains the ejb timer service beans.
`ejb3-connectors-jboss-beans.xml`	`ejb3-connectors-jboss-beans.xml` EJB3 remoting transport beans.
`ejb3-container-jboss-beans.xml`	`ejb3-container-jboss-beans.xml` UserTransaction integration bean for the EJB3 containers.
`ejb3-interceptors-aop.xml`	`ejb3-interceptors-aop.xml` defines the EJB3 container aspects.
`ejb3-timer-service.xml`	`ejb3-timer-service.xml` an alternate quartz based timer service
`hdscanner-jboss-beans.xml`	`hdscanner-jboss-beans.xml` the deploy directory hot deployment scanning bean
hsqldb-ds.xml	configures the Hypersonic embedded database service configuration file. It sets up the embedded database and related connection factories.
http-invoker.sar	

File	Description
	contains the detached invoker that supports RMI over HTTP. It also contains the proxy bindings for accessing JNDI over HTTP.
jboss-local-jdbc.rar	is a JCA resource adaptor that implements the JCA `ManagedConnectionFactory` interface for JDBC drivers that support the `DataSource` interface but not JCA.
jboss-xa-jdbc.rar	JCA resource adaptors for XA DataSources
`jbossweb.sar`	an mbean service supporting TomcatDeployer with web application deployment service management.
`jbossws.sar`	provides JEE web services support.
`jca-jboss-beans.xml`	`jca-jboss-beans.xml` is the application server implementation of the JCA specification. It provides the connection management facilities for integrating resource adaptors into the JBoss server.
`jms-ra.rar`	`jms-ra.rar` JBoss JMS Resource Adapter
`messaging/connection-factories-service.xml`	configures the DLQ, ExpiryQueue JMS connection factory
`messaging/destinations-service.xml`	The message persistence store service
`messaging/destinations-service.xml`	configures the DLQ, ExpiryQueue JMS destinations.
`messaging/jms-ds.xml`	`jms-ds.xml` configures the JMSProviderLoader and JmsXA inflow resource adaptor connection factory binding.
`messaging/legacy-service.xml`	`legacy-service.xml` configures the JMSProviderLoader and JmsXA inflow resource adaptor connection factory binding.
`messaging/messaging-jboss-beans.xml`	The `messaging-jboss-beans.xml` file configures JMS security and management beans.
`messaging/messaging-service.xml`	The `messaging-service.xml` file configures the core JBoss Messaging service.
`messaging/remoting-bisocket-service.xml`	The `remoting-bisocket-service.xml` configures the JMS remoting service layer.
`jmx-console.war`	The `jmx-console.war` directory provides the JMX Console. The JMX Console provides a

File	Description
	simple web interface for managing the MBean server.
`jmx-invoker-service.xml`	`jmx-invoker-service.xml` is an MBean service archive that exposes a subset of the JMX `MBeanServer` interface methods as an RMI interface to enable remote access to the JMX core functionality.
`jmx-remoting.sar`	`jmx-remoting.sar` is a javax.management.remote implementation providing access to the JMX server.
`legacy-invokers-service.xml`	`legacy-invokers-service.xml` the legacy detached jmx invoker remoting services.
`jsr-88-service.xml`	`jsr-88-service.xml` provides the JSR 88 remote deployment service.
`mail-ra.rar`	`mail-ra.rar` is a resource adaptor that provides a JavaMail connector.
`mail-service.xml`	The `mail-service.xml` file is an MBean service descriptor that provides JavaMail sessions for use inside the JBoss server.
`monitoring-service.xml`	The `monitoring-service.xml` file configures alert monitors like the console listener and email listener used by JMX notifications.
`profileservice-jboss-beans.xml`	`profileservice-jboss-beans.xml` description
`properties-service.xml`	The `properties-service.xml` file is an MBean service descriptor that allows for customization of the JavaBeans `PropertyEditors` as well as the definition of system properties.
`quartz-ra.rar`	`quartz-ra.rar` is a resource adaptor for inflow of Quartz events
`remoting-jboss-beans.xml`	`remoting-jboss-beans.xml` contains the unified invokers based on JBoss Remoting.
`scheduler-service.xml`	The `scheduler-service.xml` and `schedule-manager-service.xml` files are MBean service descriptors that provide a scheduling type of service.
`security/security-jboss-beans.xml`	`security-jboss-beans.xml` security domain related beans.

File	Description
`security/security-policies-jboss-beans.xml`	`security-policies-jboss-beans.xml` security authorization related beans for ejb and web authorization.
`sqlexception-service.xml`	The `sqlexception-service.xml` file is an MBean service descriptor for the handling of vendor specific `SQLExceptions`.
`transaction-jboss-beans.xml`	`transaction-jboss-beans.xml` JTA transaction manager related beans.
`transaction-service.xml`	`transaction-service.xml` ClientUserTransaction proxy service configuration.
`uuid-key-generator.sar`	The `uuid-key-generator.sar` service provides a UUID-based key generation facility.

9.2.3. The "all" Server Configuration File Set

The "all" server configuration file set is located in the `<JBoss_Home>/server/all` directory. In addition to the services in the "default" set, the all configuration contains several other services in the `conf/` directory as shown below.

Table 9.5. Additional Services in "conf" directory for "all" configuration

File	Description
`cluster-service.xml`	This service configures clustering communication for most clustered services in JBoss.
`deploy-hasingleton-service.xml`	This provides the HA singleton service, allowing JBoss to manage services that must be active on only one node of a cluster.
`httpha-invoker.sar`	This service provides HTTP tunneling support for clustered environments.
`iiop-service.xml`	This provides IIOP invocation support.
`juddi-service.sar`	This service provides UDDI lookup services.
`snmp-adaptor.sar`	This is a JMX to SNMP adaptor. It allows for the mapping of JMX notifications onto SNMP traps.

9.2.4. EJB3 Services

The following table explains the files providing ejb3 services.

Table 9.6. EJB3 Services

File	Description
`ejb3-interceptors-aop.xml`	This service provides the AOP interceptor stack configurations for EJB3 bean types.
`ejb3.deployer`	This service deploys EJB3 applications into JBoss.
`jboss-aop-jdk50.deployer`	This is a Java 5 version of the AOP deployer. The AOP deployer configures the `AspectManagerService` and deploys JBoss AOP applications.
`jbossws.sar`	This provides Java EE 5 web services support.

Finally, in the EJB3 "all" configuration there are two additional services.

Table 9.7. Additional Services in EJB3 "all" Configuration

File	Description
`ejb3-clustered-sfsbcache-service.xml`	This provides replication and failover for EJB3 stateful session beans.
`ejb3-entity-cache-service.xml`	This provides a clustered cache for EJB3 entity beans.

9.2.5.

9.3. Starting and Stopping the Server

9.3.1. Start the Server

Move to `JBOSS_DIST/jboss-as/bin` directory and execute the `run.bat` (for Windows) or `run.sh` (for Linux) script, as appropriate for your operating system.

Remote connection to the JBoss AS server

JBoss AS now binds its services to localhost (127.0.0.1) by default, instead of binding to all available interfaces (0.0.0.0). This was primarily done for security reasons because of concerns of users going to production without having secured their servers properly. To enable remote access by binding JBoss services to a particular interface, simply run jboss with the `-b` option. To bind to all available interfaces and re-enable the legacy behaviour use `-b 0.0.0.0`. In any case, be aware you still need to secure your server properly.

For more information including setting up multiple JBoss server instances on one machine and hosting multiple domains with JBoss, please refer to the *Administration and*

Configuration Guide [http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Server_Configuration_Guide/beta500/html-single/index.html]. Some examples on binding are shipped in `<JBOSS_HOME>/docs/examples/binding-manager/sample-bindings.xml`.

On starting your server, your screen output should look like the following (accounting for installation directory differences) and contain no error or exception messages:

```
[user@mypc bin]$ ./run.sh
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /home/user/jboss-as-version/jboss-as

  JAVA: java

  JAVA_OPTS: -Dprogram.name=run.sh -server -Xms1503m -Xmx1503m -Dsun.rmi.dgc.client.
gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Djava.net.preferIPv4Stack=true

  CLASSPATH: /home/user/jboss-as-version/jboss-as/bin/run.jar

=========================================================================
```

More options for the JBoss AS `run` script are discussed in *Section 9.3.2, “Start the Server With Alternate Configuration”* below.

Note

Note that there is no "Server Started" message shown at the console when the server is started using the `production` profile, which is the default profile used when no other is specified. This message may be observed in the `server.log` file located in the `server/production/log` subdirectory.

9.3.2. Start the Server With Alternate Configuration

Using `run.sh` without any arguments starts the server using the `default` server configuration file set. To start with an alternate configuration file set, pass the name of the server configuration file set [same as the name of the server configuration directory under `JBOSS_DIST/jboss-as/server`] that you want to use, as the value to the `-c` command line option. For example, to start with the `minimal` configuration file set you should specify:

```
[bin]$ ./run.sh -c minimal
...
...
...
15:05:40,301 INFO  [Server] JBoss (MX MicroKernel) [5.0.0 (build: SVNTag=JBoss_5_0_0
 date=200801092200)] Started in 5s:75ms
```

9.3.3. Using run.sh

The `run` script supports the following options:

```


usage: run.sh [options]
-h, --help                           Show help message
-V, --version                        Show version information
--                                   Stop processing options
-D<name>[=<value>]       Set a system property
-d, --bootdir=<dir>          Set the boot patch directory; Must be absolute or url
-p, --patchdir=<dir>         Set the patch directory; Must be absolute or url
-n, --netboot=<url>          Boot from net with the given url as base
-c, --configuration=<name>    Set the server configuration name
-B, --bootlib=<filename>      Add an extra library to the front bootclasspath
-L, --library=<filename>      Add an extra library to the loaders classpath
-C, --classpath=<url>         Add an extra url to the loaders classpath
-P, --properties=<url>        Load system properties from the given url
-b, --host=<host or ip>       Bind address for all JBoss services.
-g, --partition=<name>        HA Partition name (default=DefaultDomain)
-u, --udp=<ip>               UDP multicast address
-l, --log=<log4j|jdk>         Specify the logger plugin type
```

9.3.4. Stopping the Server

To shutdown the server, you simply issue a Ctrl-C sequence in the console in which JBoss was started. Alternatively, you can use the `shutdown.sh` command.

```
[bin]$ ./shutdown.sh -S
```

The `shutdown` script supports the following options:

```
A JMX client to shutdown (exit or halt) a remote JBoss server.
```

```
usage: shutdown [options] <operation>

options:
-h, --help                Show this help message (default)
-D<name>[=<value>]        Set a system property
--                        Stop processing options
-s, --server=<url>        Specify the JNDI URL of the remote server
-n, --serverName=<url>    Specify the JMX name of the ServerImpl
-a, --adapter=<name>      Specify JNDI name of the MBeanServerConnection to use
-u, --user=<name>         Specify the username for authentication
-p, --password=<name>     Specify the password for authentication

operations:
-S, --shutdown            Shutdown the server
-e, --exit=<code>         Force the VM to exit with a status code
-H, --halt=<code>         Force the VM to halt with a status code
```

Using the shutdown command requires a server configuration that contains the `jmx-invoker-service.xml` service. Hence you cannot use the shutdown command with the `minimal` configuration.

9.3.5. Running as a Service under Microsoft Windows

You can configure the server to run as a service under Microsoft Windows, and configure it to start automatically if desired.

Download the `JavaService` package from *http://forge.objectweb.org/projects/javaservice/*.

Unzip the package and use the `JBossInstall.bat` file to install the JBoss service. You must set the `JAVA_HOME` and `JBOSS_HOME` environment variables to point to the `jdk` and `jboss-as` directories before running `JBossInstall.bat`. Run `JBossInstall.bat` with the following syntax:

```
JBossInstall.bat <depends> [-auto | -manual]
```

Where *<depends>* is the name of any service that the JBoss AS server depends on, such as the `mysql` database service.

Once the service is installed the server can be started by using the command `net start JBoss`, and stopped with the command `net stop JBoss`.

Please refer to the documentation included in the `JavaService` package for further information.

9.4. The JMX Console

When the JBoss Server is running, you can get a live view of the server by going to the JMX console application at *http://localhost:8080/jmx-console*. You should see something similar to *Figure 9.1, "View of the JMX Management Console Web Application"*.

The JMX Console is the JBoss Management Console which provides a raw view of the JMX MBeans which make up the server. They can provide a lot of information about the running server and allow you to modify its configuration, start and stop components and so on.

For example, find the `service=JNDIView` link and click on it. This particular MBean provides a service to allow you to view the structure of the JNDI namespaces within the server. Now find the operation called `list` near the bottom of the MBean view page and click the `invoke` button. The operation returns a view of the current names bound into the JNDI tree, which is very useful when you start deploying your own applications and want to know why you can't resolve a particular EJB name.

Figure 9.1. View of the JMX Management Console Web Application

Look at some of the other MBeans and their listed operations; try changing some of the configuration attributes and see what happens. With a very few exceptions, none of the changes made through the console are persistent. The original configuration will be reloaded when you restart JBoss, so you can experiment freely without doing any permanent damage.

Note

If you installed JBoss using the graphical installer, the JMX Console will prompt you for a username and password before you can access it. If you installed using other modes, you can still configure JMX Security manually. We will show you how to secure your console in *Section 9.6.4, "Security Service"*.

9.5. Hot-deployment of services in JBoss

Hot-deployable services are those which can be added to or removed from the running server. These are placed in the `JBOSS_DIST/jboss-as/server/<instance-name>/deploy` directory. Let's have a look at a practical example of hot-deployment of services in JBoss before we go on to look at server configuration issues in more detail.

Start JBoss if it isn't already running and take a look at the `server/production/deploy` directory. Remove the `mail-service.xml` file and watch the output from the server:

```
13:10:05,235 INFO  [MailService] Mail service 'java:/Mail' removed from JNDI
```

Then replace the file and watch JBoss re-install the service:

```
13:58:54,331 INFO  [MailService] Mail Service bound to java:/Mail
```

This is hot-deployment in action.

9.6. Basic Configuration Issues

Now that we have examined the JBoss server, we will take a look at some of the main configuration files and what they are used for. All paths are relative to the server configuration directory (`server/default`, for example).

9.6.1. Bootstrap Configuration

The microcontainer bootstrap configuration is described by the `conf/bootstrap.xml` and the `conf/bootstrap/*.xml` it references. Its expected that the number of bootstrap beans will be reduced in the future. Its not expected that you would need to edit the bootstrap configuration files for a typical installation.

9.6.2. Legacy Core Services

The legacy core services specified in the `conf/jboss-service.xml` file are started just after server starts up the microcontainer. If you have a look at this file in an editor you will see MBeans for various services including logging, security, JNDI, JNDIView etc. Try commenting out the entry for the `JNDIView` service.

> **Note**
>
> Eventually this file will be dropped as the services are converted to microcontainer beans or mbeans that are deployed as deploy directory services.

Note that because the mbeans definition had nested comments, we had to comment out the mbean in two sections, leaving the original comment as it was.

```
<!-- Section 1 commented out
<mbean code="org.jboss.naming.JNDIView"
   name="jboss:service=JNDIView"
   xmbean-dd="resource:xmdesc/JNDIView-xmbean.xml">
-->
   <!-- The HANamingService service name -->
<!-- Section two commented out
   <attribute name="HANamingService">jboss:service=HAJNDI</attribute></mbean>
```

```
-->
```

If you then restart JBoss, you will see that the `JNDIView` service no longer appears in the JMX Management Console (JMX Console) listing. In practice, you should rarely, if ever, need to modify this file, though there is nothing to stop you adding extra MBean entries in here if you want to. The alternative is to use a separate file in the `deploy` directory, which allows your service to be hot deployable.

9.6.3. Logging Service

In JBoss `log4j` is used for logging. If you are not familiar with the `log4j` package and would like to use it in your applications, you can read more about it at the Jakarta web site (*http://jakarta.apache.org/log4j/*).

Logging is controlled from a central `conf/jboss-log4j.xml` file. This file defines a set of appenders specifying the log files, what categories of messages should go there, the message format and the level of filtering. By default, JBoss produces output to both the console and a log file (`log/server.log`).

There are 6 basic log levels used: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR` and `FATAL`. The logging threshold on the console is `INFO`, which means that you will see informational messages, warning messages and error messages on the console but not general debug messages. In contrast, there is no threshold set for the `server.log` file, so all generated logging messages will be logged there.

If things are going wrong and there doesn’t seem to be any useful information in the console, always check the `server.log` file to see if there are any debug messages which might help you to track down the problem. However, be aware that just because the logging threshold allows debug messages to be displayed, that doesn't mean that all of JBoss will produce detailed debug information for the log file. You will also have to boost the logging limits set for individual categories. Take the following category for example.

```
<!-- Limit JBoss categories to INFO -->
<category name="org.jboss">
   <priority value="INFO"/>
</category>
```

This limits the level of logging to `INFO` for all JBoss classes, apart from those which have more specific overrides provided. If you were to change this to `DEBUG`, it would produce much more detailed logging output.

As another example, let’s say you wanted to set the output from the container-managed persistence engine to `DEBUG` level and to redirect it to a separate file, `cmp.log`, in order to analyze the generated SQL commands. You would add the following code to the `conf/jboss-log4j.xml` file:

```
<appender name="CMP" class="org.jboss.logging.appender.RollingFileAppender">
   <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
   <param name="File" value="${jboss.server.home.dir}/log/cmp.log"/>
   <param name="Append" value="false"/>
   <param name="MaxFileSize" value="500KB"/>
   <param name="MaxBackupIndex" value="1"/>

   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
   </layout>
</appender>

<category name="org.jboss.ejb.plugins.cmp">
   <priority value="DEBUG" />
   <appender-ref ref="CMP"/>
</category>
```

This creates a new file appender and specifies that it should be used by the logger (or category) for the package `org.jboss.ejb.plugins.cmp`.

The file appender is set up to produce a new log file every day rather than producing a new one every time you restart the server or writing to a single file indefinitely. The current log file is `cmp.log`. Older files have the date they were written added to their filenames. Please note that the `log` directory also contains HTTP request logs which are produced by the web container.

9.6.4. Security Service

The security domain information is stored in the file `conf/login-config.xml` as a list of named security domains, each of which specifies a number of JAAS [1] login modules which are used for authentication purposes in that domain. When you want to use security in an application, you specify the name of the domain you want to use in the application's JBoss-specific deployment descriptors, `jboss.xml` (used in defining jboss specific configurations for an application) and/or `jboss-web.xml` (used in defining jboss for a Web application. We'll quickly look at how to do this to secure the JMX Console application which ships with JBoss.

Almost every aspect of the JBoss server can be controlled through the JMX Console, so it is important to make sure that, at the very least, the application is password protected. Otherwise, any remote user could completely control your server. To protect it, we will add a security domain to cover the application. This can be done in the `jboss-web.xml` file for the JMX Console, which can be found in `deploy/jmx-console.war/WEB-INF/` directory. Uncomment the `security-domain` in that file, as shown below.

[1] The Java Authentication and Authorization Service. JBoss uses JAAS to provide pluggable authentication modules. You can use the ones that are provided or write your own if you have more specific requirements.

```
<jboss-web>
    <security-domain>java:/jaas/jmx-console</security-domain>
</jboss-web>
```

This links the security domain to the web application, but it doesn't tell the web application what security policy to enforce, what URLs are we trying to protect, and who is allowed to access them. To configure this, go to the `web.xml` file in the same directory and uncomment the `security-constraint` that is already there. This security constraint will require a valid user name and password for a user in the `JBossAdmin` group.

```
<!--
   A security constraint that restricts access to the HTML JMX console
   to users with the role JBossAdmin. Edit the roles to what you want and
   uncomment the WEB-INF/jboss-web.xml/security-domain element to enable
   secured access to the HTML JMX console.
-->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HtmlAdaptor</web-resource-name>
        <description>
            An example security config that only allows users with the
            role JBossAdmin to access the HTML JMX console web application
        </description>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>JBossAdmin</role-name>
    </auth-constraint>
</security-constraint>
```

That's great, but where do the user names and passwords come from? They come from the `jmx-console` security domain we linked the application to. We have provided the configuration for this in the `conf/login-config.xml`.

```
<application-policy name="jmx-console">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
                  flag="required">
            <module-option name="usersProperties">
```

```
                props/jmx-console-users.properties
            </module-option>
            <module-option name="rolesProperties">
                props/jmx-console-roles.properties
            </module-option>
        </login-module>
    </authentication>
</application-policy>
```

This configuration uses a simple file based security policy. The configuration files are found in the `conf/props` directory of your server configuration. The usernames and passwords are stored in the `conf/props/jmx-console-users.properties` file and take the form "`username=password`". To assign a user to the `JBossAdmin` group add "`username=JBossAdmin`" to the `jmx-console-roles.properties` file (additional roles on that username can be added comma separated). The existing file creates an `admin` user with the password `admin`. For security, please either remove the user or change the password to a stronger one.

JBoss will re-deploy the JMX Console whenever you update its `web.xml`. You can check the server console to verify that JBoss has seen your changes. If you have configured everything correctly and re-deployed the application, the next time you try to access the JMX Console, it will ask you for a name and password. [2]

The JMX Console isn't the only web based management interface to JBoss. There is also the Web Console. Although it's a Java applet, the corresponding web application can be secured in the same way as the JMX Console. The Web Console is in the file `deploy/management/console-mgr.sar/web-console.war`.. The only difference is that the Web Console is provided as a simple WAR file instead of using the exploded directory structure that the JMX Console did. The only real difference between the two is that editing the files inside the WAR file is a bit more cumbersome.

9.6.5. Additional Services

The non-core, hot-deployable services are added to the `deploy` directory. They can be either XML descriptor files, `*-service.xml, *-jboss-beans.xml`, MC `.beans` archive, or JBoss Service Archive (SAR) files. SARs contains an META-INF/jboss-service.xml descriptor and additional resources the service requires (e.g. classes, library JAR files or other archives), all packaged up into a single archive. Similarly, a `.beans` archive contains a META-INF/jboss-beans.xml and additional resources.

Detailed information on all these services can be found in the *JBoss Application Server: Configuration Guide*, which also provides comprehensive information on server internals and the implementation of services such as JTA and the J2EE Connector Architecture (JCA).

[2] Since the username and password are session variables in the web browser you may need to restart your browser to use the login dialog window.

EJB3 Caveats in JBoss Application Server 5.0.0

There are a number of implementation features that you should be aware of when developing applications for JBoss Application Server 5.0.0.

10.1. Unimplemented features

The Release Notes for JBoss Application Server contain information on EJB3 features that are not yet implemented, or partially implemented. The Release Notes include links to issues in JIRA for information on workarounds and further details.

10.2. Referencing EJB3 Session Beans from non-EJB3 Beans

JBoss Application Server 5 fully supports the entire Java 5 Enterprise Edition specification. JBoss Application Server 4.2.2 implemented EJB3 functionality by way of an EJB MBean container running as a plugin in the JBoss Application Server. This had certain implications for application development.

The EJB3 plugin injects references to an EntityManager and @EJB references from one EJB object to another. However this support is limited to the EJB3 MBean and the JAR files it manages. Any JAR files which are loaded from a WAR (such as Servlets, JSF backing beans, and so forth) do not undergo this processing. The Java 5 Enterprise Edition standard specifies that a Servlet can reference a Session Bean through an @EJB annotated reference, this was not implemented in JBoss Application Server 4.2.2.

Sample Applications

The JBoss Application Server, ships with various sample applications under `JBOSS_HOME/docs/examples`.

For further details, please refer to the accompanying `readme.txt` for the respective sample applications under the above directory.

Sample JSF-EJB3 Application

We use a simple "TODO" application to show how JSF and EJB3 work together in a web application. The "TODO" application works like this: You can create a new 'todo' task item using the "Create" web form. Each 'todo' item has a 'title' and a 'description'. When you submit the form, the application saves your task to a relational database. Using the application, you can view all 'todo' items, edit/delete an existing 'todo' item and update the task in the database.

The sample application comprises the following components:

- Entity objects - These objects represent the data model; the properties in the object are mapped to column values in relational database tables.
- JSF web pages - The web interface used to capture input data and display result data. The data fields on these web pages are mapped to the data model via the JSF Expression Language (EL).
- EJB3 Session Bean - This is where the functionality is implemented. We make use of a Stateless Session Bean.

12.1. Data Model

Let's take a look at the contents of the Data Model represented by the `Todo` class in the `Todo.java` file. Each instance of the `Todo` class corresponds to a row in the relational database table. The 'Todo' class has three properties: id, title and description. Each of these correspond to a column in the database table.

The 'Entity class' to 'Database Table' mapping information is specified using EJB3 Annotations in the 'Todo' class. This eliminates the need for XML configuration and makes it a lot clearer. The `@Entity` annotation defines the `Todo` class as an Entity Bean. The `@Id` and `@GeneratedValue` annotations on the `id` property indicate that the `id` column is the primary key and that the server automatically generates its value for each `Todo` object saved into the database.

```
@Entity
public class Todo implements Serializable {

  private long id;
  private String title;
  private String description;

  public Todo () {
    title ="";
    description ="";
  }
```

```
 @Id @GeneratedValue
 public long getId() { return id;}
 public void setId(long id) { this.id = id; }

 public String getTitle() { return title; }
 public void setTitle(String title) {this.title = title;}

 public String getDescription() { return description; }
 public void setDescription(String description) {
   this.description = description;
 }

}
```

12.2. JSF Web Pages

In this section we will show you how the web interface is defined using JSF pages. We will also see how the data model is mapped to the web form using JSF EL. Using the #{...} notation to reference Java objects is called **JSF EL** (JSF Expression Language). Lets take a look at the pages used in our application:

- **index.xhtml**: This page displays two options: 1. Create New Todo 2. Show all Todos. When you click on the Submit button the corresponding action is invoked.

```


<h:form>
<ul>
  <li><h:commandLink type="submit" value="Create New Todo" action="create"/></li>
  <li><h:commandLink type="submit" value="Show All Todos" action="todos"/></li>
</ul>
</h:form>
```

- **create.xhtml**: When you try to create a new task, this JSF page captures the input data. We use the `todoBean` to back the form input text fields. The #{todoBean.todo.title} symbol refers to the "title" property of the "todo" object in the "TodoBean" class. The #{todoBean.todo.description} symbol refers to the "description" property of the "todo" object in the "TodoBean" class. The #{todoBean.persist} symbol refers to the "persist" method in the "TodoBean" class. This method creates the "Todo" instance with the input data (title and description) and persists the data.

```


<h:form id="create">
```

```
<table>
  <tr>
    <td>Title:</td>
    <td>
      <h:inputText id="title" value="#{todoBean.todo.title}" size="15">
        <f:validateLength minimum="2"/>
      </h:inputText>
    </td>
  </tr>
  <tr>
    <td>Description:</td>
    <td>
      <h:inputTextarea id="description" value="#{todoBean.todo.description}">
        <f:validateLength minimum="2" maximum="250"/>
      </h:inputTextarea>
    </td>
  </tr>
</table>
<h:commandButton type="submit" id="create" value="Create"
                 action="#{todoBean.persist}"/>
</h:form>
```

Figure 12.1, “The "Create Todo" web page ” shows the "Create Todo" web page with the input fields mapped to the data model.

Figure 12.1. The "Create Todo" web page

- **todos.xhtml**: This page displays the list of all "todos" created. There is also an option to choose a "todo" item for 'edit' or 'delete'.

  The list of all 'todos' is fetched by #{todoBean.todos} symbol referring to the 'getTodos()' property in the 'TodoBean' class. The JSF `dataTable` iterates through the list and displays each `Todo` object in a row. The 'Edit' option is available across each row. The #{todo.id} symbol represents the "id" property of the "todo" object.

```
<h:form>
<h:dataTable value="#{todoBean.todos}" var="todo">
  <h:column>
    <f:facet name="header">Title</f:facet>
    #{todo.title}
  </h:column>
  <h:column>
    <f:facet name="header">Description</f:facet>
    #{todo.description}
  </h:column>
  <h:column>
    <a href="edit.faces?tid=#{todo.id}">Edit</a>
  </h:column>
</h:dataTable>
<center>
  <h:commandButton action="create"
            value="Create New Todo" type="submit"/>
</center>
</h:form>
```

Figure 12.2, “The "Show All Todos" web page ” shows the "Show All Todos" web page with the data fields mapped to the data model.

Figure 12.2. The "Show All Todos" web page

- **edit.xhtml**: This page allows you to edit the "todo" item's 'title' and 'description' properties. The #{todoBean.update} and #{todoBean.delete} symbols represent the "update" and "delete" methods in the "TodoBean" class.

```
<h2>Edit #{todoBean.todo.title}</h2>
<h:form id="edit">
<input type="hidden" name="tid" value="#{todoBean.todo.id}"/>
<table>
  <tr>
    <td>Title:</td>
    <td>
      <h:inputText id="title" value="#{todoBean.todo.title}" size="15">
        <f:validateLength minimum="2"/>
      </h:inputText>
    </td>
  </tr>
  <tr>
    <td>Description:</td>
    <td>
      <h:inputTextarea id="description" value="#{todoBean.todo.description}">
        <f:validateLength minimum="2" maximum="250"/>
      </h:inputTextarea>
    </td>
  </tr>
</table>
<h:commandButton type="submit" id="update" value="Update"
                 action="#{todoBean.update}"/>
<h:commandButton type="submit" id="delete" value="Delete"
                 action="#{todoBean.delete}"/>
</h:form>
```

Figure 12.3, “The "Edit Todo" web page ” shows the "Edit Todo" web page with the mapping to the data model.

Figure 12.3. The "Edit Todo" web page

Note

We have used XHTML pages in the sample applications because we recommend using Facelets instead of JSP to render JSF view pages.

12.3. EJB3 Session Beans

EJB 3.0 is one of the major improvements introduced with Java EE 5.0. It aims at reducing the complexity of older versions of EJB and simplifies Enterprise Java development and deployment. You will notice that to declare a class as a 'Session Bean' you simply have to annotate it. Using annotations eliminates the complexity involved with too many deployment descriptors. Also the only interface an EJB3 Session Bean requires is a business interface that declares all the business methods that must be implemented by the bean.

We will explore the two important source files associated with the Bean implementation in our application: `TodoDaoInt.java` and `TodoDao.java`.

- **Business interface**: `TodoDaoInt.java`

  We define here the methods that need to be implemented by the bean implementation class. Basically, the business methods that will be used in our application are defined here.

```
public interface TodoDaoInt {

  public void persist (Todo todo);
  public void delete (Todo todo);
  public void update (Todo todo);

  public List <Todo> findTodos ();
  public Todo findTodo (String id);
}
```

- **Stateless Session Bean**: `TodoDao.java`

  The `@Stateless` annotation marks the bean as a stateless session bean. In this class, we need to access the Entity bean `Todo` defined earlier. For this we need an `EntityManager`. The `@PersistenceContext` annotation tells the JBoss Server to inject an entity manager during deployment.

```


@Stateless
public class TodoDao implements TodoDaoInt {

  @PersistenceContext
  private EntityManager em;

  public void persist (Todo todo) {
    em.persist (todo);
  }

  public void delete (Todo todo) {
    Todo t = em.merge (todo);
    em.remove( t );
  }

  public void update (Todo todo) {
    em.merge (todo);
  }

  public List <Todo> findTodos () {
    return (List <Todo>) em.createQuery("select t from Todo t")
                                  .getResultList();
  }

  public Todo findTodo (String id) {
    return (Todo) em.find(Todo.class, Long.parseLong(id));
  }

}
```

12.4. Configuration and Packaging

We will build the sample application using Ant and explore the configuration and packaging details. Please install Ant if currently not installed on your computer.

12.4.1. Building The Application

Let's look at building the example application and then explore the configuration files in detail.

In *Chapter 11, Sample Applications*, we looked at the directory structure of the `jsfejb3` sample application. At the command line, go to the `jsfejb3` directory. There you will see a `build.xml` file. This is our Ant build script for compiling and packaging the archives. To build the application, you need to first of all edit the `build.xml` file and edit the value of `jboss-dist` to reflect the location where the JBoss Application Server is installed. Once you have done this, just type the command `ant` and your output should look like this:

```
[user@localhost jsfejb3]$ ant
Buildfile: build.xml

compile:
    [mkdir] Created dir: /jboss/gettingstarted/jsfejb3/build/classes
    [javac] Compiling 4 source files to
 /home/user/Desktop/gettingstarted/jsfejb3/build/classes
    [javac] Note: /jboss/gettingstarted/jsfejb3/src/TodoDao.java uses
 unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

war:
    [mkdir] Created dir: /jboss/gettingstarted/jsfejb3/build/jars
      [war] Building war: /jboss/gettingstarted/jsfejb3/build/jars/app.war

ejb3jar:
      [jar] Building jar: /jboss/gettingstarted/jsfejb3/build/jars/app.jar

ear:
      [ear] Building ear:
 /jboss/gettingstarted/jsfejb3/build/jars/jsfejb3.ear

main:

BUILD SUCCESSFUL
Total time: 3 seconds
```

If you get the BUILD SUCCESSFUL message, you will find a newly created `build` directory with 2 sub-directories in it:

- **classes**: containing the compiled class files.

- **jars**: containing three archives - `app.jar`, `app.war` and `jsfejb3.ear`.
  - app.jar : EJB code and descriptors.
  - app.war : web application which provides the front end to allow users to interact with the business components (the EJBs). The web source (HTML, images etc.) contained in the `jsfejb3/view` directory is added unmodified to this archive. The Ant task also adds the `WEB-INF` directory that contains the files which aren't meant to be directly accessed by a web browser but are still part of the web application. These include the deployment descriptors (`web.xml`) and extra jars required by the web application.
  - jsfejb3.ear : The EAR file is the complete application, containing the EJB modules and the web module. It also contains an additional descriptor, `application.xml`. It is also possible to deploy EJBs and web application modules individually but the EAR provides a convenient single unit.

12.4.2. Configuration Files

Now that we have built the application, lets take a closer look at some of the important Configuration files. We have built the final archive ready for deployment - `jsfejb3.ear`. The contents of your EAR file should look like this:

```
jsfejb3.ear
|+ app.jar   // contains the EJB code
    |+ import.sql
    |+ Todo.class
    |+ TodoDao.class
    |+ TodoDaoInt.class
    |+ META-INF
        |+ persistence.xml
|+ app.war   // contains web UI
    |+ index.html
    |+ index.xhtml
    |+ create.xhtml
    |+ edit.xhtml
    |+ todos.xhtml
    |+ TodoBean.class
    |+ style.css
    |+ META-INF
    |+ WEB-INF
       |+ faces-config.xml
       |+ navigation.xml
       |+ web.xml
|+ META-INF  // contains the descriptors
```

```
  |+ application.xml
  |+ jboss-app.xml
```

- `application.xml`: This file lists the JAR files in the EAR (in our case `app.jar`) and tells the JBoss server what files to look for and where. The root URL for the application is also specified in this file as 'context-root'.

```
<application>
  <display-name>Sample Todo</display-name>
  <module>
    <web>
      <web-uri>app.war</web-uri>
      <context-root>/jsfejb3</context-root>
    </web>
  </module>
  <module>
    <ejb>app.jar</ejb>
  </module>
</application>
```

- `jboss-app.xml`: Every EAR application should specify a unique string name for the class loader. In our case, we use the application name 'jsfejb3' as the class loader name.

```
<jboss-app>
  <loader-repository>
    jsfejb3:archive=jsfejb3.ear
  </loader-repository>
</jboss-app>
```

- `app.jar`: This contains EJB3 Session Bean and Entity Bean classes and the related configuration files. In addition, the `persistence.xml` file configures the back-end data source (in our case the default HSQL database) for the `EntityManager`.

```
<persistence>
   <persistence-unit name="helloworld">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:/DefaultDS</jta-data-source>
      <properties>
        <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
        <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      </properties>
    </persistence-unit>
  </persistence>
```

- `app.war`: This contains the Web UI files packaged according to the Web Application aRchive (WAR) specification. It contains all the web pages and the required configuration files. The `web.xml` file is an important file for all JAVA EE web applications. It is the web deployment descriptor file. The `faces-config.xml` file is the configuration file for JSF. The `navigation.xml` file contains the rules for JSF page navigation.

```


//faces-config.xml
<faces-config>
  <application>
    <view-handler>
      com.sun.facelets.FaceletViewHandler
    </view-handler>
  </application>
  <managed-bean>
    <description>Dao</description>
    <managed-bean-name>todoBean</managed-bean-name>
    <managed-bean-class>TodoBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
</faces-config>
```

12.5. The Database

12.5.1. Creating the Database Schema

To pre-populate the database, we have supplied SQL Code (`import.sql`) to run with HSQL in the `examples/jsfejb3/resources` directory. When you build the application using Ant, this is packaged in the app.jar file within the jsfejb3.ear file. When the application is deployed, you should be able to view the pre-populated data.

12.5.2. The HSQL Database Manager Tool

Just as a quick aside at this point, start up the JMX console application and click on the `service=Hypersonic` link which you'll find under the section `jboss`. If you can't find this, make sure the Hypersonic service is enabled in the `hsqldb-ds.xml` file.

This will take you to the information for the Hypersonic service MBean. Scroll down to the bottom of the page and click the `invoke` button for the `startDatabaseManager()` operation. This starts up the HSQL Manager, a Java GUI application which you can use to manipulate the database directly.

Figure 12.4. The HSQL Database Manger

12.6. Deploying the Application

Deploying an application in JBoss is simple and easy. You just have to copy the EAR file to the `deploy` directory in the 'server configuration' directory of your choice. Here, we will deploy it to the 'default' configuration, so we copy the EAR file to the `JBOSS_DIST/jboss-as/server/default/deploy` directory.

You should see something close to the following output from the server:

```
15:32:23,997 INFO  [EARDeployer] Init J2EE application:
 file:/jboss/jboss-as-5.0.0<release>/server/default/deploy/jsfejb3.ear
15:32:24,212 INFO  [JmxKernelAbstraction] creating wrapper delegate for:
 org.jboss.ejb3.
entity.PersistenceUnitDeployment
15:32:24,213 INFO  [JmxKernelAbstraction] installing MBean:
 persistence.units:ear=
jsfejb3.ear,jar=app.jar,unitName=helloworld with dependencies:
15:32:24,213 INFO  [JmxKernelAbstraction]
 jboss.jca:name=DefaultDS,service=
DataSourceBinding
15:32:24,275 INFO  [PersistenceUnitDeployment] Starting persistence unit
 persistence.
units:ear=jsfejb3.ear,jar=app.jar,unitName=helloworld
15:32:24,392 INFO  [Ejb3Configuration] found EJB3 Entity bean: Todo
15:32:24,450 WARN  [Ejb3Configuration] Persistence provider caller does not
 implements
the EJB3 spec correctly. PersistenceUnitInfo.getNewTempClassLoader() is
 null.
15:32:24,512 INFO  [Configuration] Reading mappings from resource :
 META-INF/orm.xml
15:32:24,512 INFO  [Ejb3Configuration] [PersistenceUnit: helloworld] no
 META-INF/orm.xml
found
15:32:24,585 INFO  [AnnotationBinder] Binding entity from annotated class:
 Todo
15:32:24,586 INFO  [EntityBinder] Bind entity Todo on table Todo
.
.
.
```

```
.
15:32:26,311 INFO  [SchemaExport] Running hbm2ddl schema export
15:32:26,312 INFO  [SchemaExport] exporting generated schema to database
15:32:26,314 INFO  [SchemaExport] Executing import script: /import.sql
15:32:26,418 INFO  [SchemaExport] schema export complete
15:32:26,454 INFO  [NamingHelper] JNDI InitialContext
 properties:{java.naming.factory.
initial=org.jnp.interfaces.NamingContextFactory,
 java.naming.factory.url.pkgs=org.jboss.
naming:org.jnp.interfaces}
15:32:26,484 INFO  [JmxKernelAbstraction] creating wrapper delegate for:
 org.jboss.ejb3.
stateless.StatelessContainer
15:32:26,485 INFO  [JmxKernelAbstraction] installing MBean:
 jboss.j2ee:ear=jsfejb3.ear,
jar=app.jar,name=TodoDao,service=EJB3 with dependencies:
15:32:26,513 INFO  [JmxKernelAbstraction]
 persistence.units:ear=jsfejb3.ear,
jar=app.jar,unitName=helloworld
15:32:26,557 INFO  [EJBContainer] STARTED EJB: TodoDao ejbName: TodoDao
15:32:26,596 INFO  [EJB3Deployer] Deployed:
 file:/jboss/jboss-as-5.0.0<release>
server/default/tmp/deploy/
tmp33761jsfejb3.ear-contents/app.jar
15:32:26,625 INFO  [TomcatDeployer] deploy, ctxPath=/jsfejb3,
 warUrl=.../tmp/deploy/
tmp33761jsfejb3.ear-contents/app-exp.war/
15:32:26,914 INFO  [EARDeployer] Started J2EE application:
 file:/jboss/jboss-as-5.0.0<release>/server/default/deploy/jsfejb3.ear
```

If there are any errors or exceptions, make a note of the error message. Check that the EAR is complete and inspect the WAR file and the EJB jar files to make sure they contain all the necessary components (classes, descriptors etc.).

You can safely redeploy the application if it is already deployed. To undeploy it you just have to remove the archive from the `deploy` directory. There's no need to restart the server in either case. If everything seems to have gone OK, then point your browser at the application URL.

http://localhost:8080/jsfejb3

You will be forwarded to the application main page. *Figure 12.5, "Sample TODO"* shows the sample application in action.

Figure 12.5. Sample TODO

Using Seam

JBoss Seam is a framework that provides the glue between the new EJB3 and JSF frameworks that are part of the Java EE 5.0 standard. In fact, the name Seam refers to the seamless manner in which it enables developers to use these two frameworks in an integrated manner. Seam automates many of the common tasks, and makes extensive use of annotations to reduce the amount of xml code that needs to be written. The overall effect is to significantly reduce the total amount of coding that needs to be done.

If you are new to Seam, you can find more introductory information from the following url and book:

- *The Seam Reference Guide* [http://docs.jboss.com/seam/2.0.0.GA/reference/en/html_single/].
- *Beginning JBoss Seam* by Joseph Faisal Nusairat, Apress 2007.

We have included two versions of the example application, one coded using EJB3 / JSF without using Seam, and one using Seam, to demonstrate clearly the difference in application development using the Seam framework.

13.1. Data Model

Let's start off our examination of the Seam implementation in the same way, by examining how the Data Model is implemented. This is done in the `Todo.java` file.

```
@Entity
@Name("todo")
public class Todo implements Serializable {

  private long id;
  private String title;
  private String description;

  public Todo () {
    title ="";
    description ="";
  }

  @Id @GeneratedValue
  public long getId() { return id;}
  public void setId(long id) { this.id = id; }

  @NotNull
  public String getTitle() { return title; }
  public void setTitle(String title) {this.title = title;}

  @NotNull
  @Length(max=250)
  public String getDescription() { return description; }
  public void setDescription(String description) {
    this.description = description;
  }

}
```

The `@Entity` annotation defines the class as an EJB3 entity bean, and tells the container to map the `Todo` class to a relational database table. Each property of the class will become a column in the table. Each instance of the class will become a row in this table. Since we have not used the `@Table` annotation, Seam's "configuration by exception" default will name the table after the class.

`@Entity` and `@Table` are both EJB3 annotations, and are not specific to Seam. It is possible to use Seam completely with POJOs (Plain Old Java Objects) without any EJB3-specific annotations. However, EJB3 brings a lot of advantages to the table, including container managed security, message-driven components, transaction and component level persistence context, and `@PersistenceContext` injection, which we will encounter a little further on.

The `@Name` annotation is specific to Seam, and defines the string name for Seam to use to register the Entity Bean. This will be the default name for the relational database table. Each component in a Seam application must have a unique name. In the other components in the Seam framework, such as JSF web pages and session beans, you can reference the managed `Todo` bean using this name. If no instance of this class exists when it is referenced from another component, then Seam will instantiate one.

The `@Id` annotation defines a primary key `id` field for the component. `@GeneratedValue` specifies that the server will automatically generate this value for the component when it is saved to the database.

Seam provides support for model-based constraints defined using Hibernate Validator, although Hibernate does not have to be the object persister used. The `@NotNull` annotation is a validation constraint that requires this property to have a value before the component can be persisted into the database. Using this annotation allows the validation to be enforced by the JSF code at the view level, without having to specify the exact validation constraint in the JSF code.

At this point the only apparent difference between the Seam version and the EJB3/JSF version of the app is the inclusion of the validator annotation `@NotNull`, and the `@Name` annotation. However, while the EJB3/JSF version of this application requires a further `TodoBean` class to be manually coded and managed in order to handle the interaction between the `Todo` class and the web interface, when using Seam the Seam framework takes care of this work for us. We'll see how this is done in practice as we examine the implementation of the user interface.

13.2. JSF Web Pages - index.xhtml and create.xhtml

The **index.xhtml** file used is the same as in the EJB3/JSF example.

create.xhtml begins to reveal the difference that coding using the Seam framework makes.

```
<h:form id="create">

<f:facet name="beforeInvalidField">
  <h:graphicImage styleClass="errorImg" value="error.png"/>
</f:facet>
<f:facet name="afterInvalidField">
  <s:message styleClass="errorMsg" />
</f:facet>
<f:facet name="aroundInvalidField">
  <s:div styleClass="error"/>
</f:facet>

<s:validateAll>

<table>

  <tr>
    <td>Title:</td>
    <td>
      <s:decorate>
        <h:inputText id="title" value="#{todo.title}" size="15"/>
      </s:decorate>
    </td>
  </tr>

  <tr>
    <td>Description:</td>
    <td>
      <s:decorate>
        <h:inputTextarea id="description" value="#{todo.description}"/>
      </s:decorate>
    </td>
  </tr>

</table>

</s:validateAll>
```

```
<h:commandButton type="submit" id="create" value="Create"
                 action="#{todoDao.persist}"/>
</h:form>
```

The first thing that is different here is the Java Server Facelet code at the beginning, which works with the `@NotNull` validation constraint of our `todo` class to enforce and indicate invalid input to the user.

Also notice here that rather than requiring the use of a `TodoBean` class as we did in the EJB3/JSF example we back the form directly with a `Todo` entity bean. When this page is called, JSF asks Seam to resolve the variable `todo` due to JSF EL references such as `#{todo.title}`. Since there is no value already bound to that variable name, Seam will instantiate an entity bean of the `todo` class and return it to JSF, after storing it in the Seam context. The Seam context replaces the need for an intermediary bean.

The form input values are validated against the Hibernate Validator constraints specified in the `todo` class. JSF will redisplay the page if the constraints are violated, or it will bind the form input values to the `Todo` entity bean.

Entity beans shouldn't do database access or transaction management, so we can't use the `Todo` entity bean as a JSF action listener. Instead, creation of a new todo item in the database is accomplished by calling the `persist` method of a `TodoDao` session bean. When JSF requests Seam to resolve the variable `todoDao` through the JSF EL expression `#{todoDao.persist}`, Seam will either instantiate an object if one does not already exist, or else pass the existing stateful `todoDao` object from the Seam context. Seam will intercept the `persist` method call and inject the `todo` entity from the session context.

Let's have a look at the `TodoDao` class (defined in `TodoDao.java`) to see how this injection capability is implemented.

13.3. Data Access using a Session Bean

Let's go through a listing of the code for the `TodoDao` class.

```
@Stateful
@Name("todoDao")
public class TodoDao implements TodoDaoInt {

  @In (required=false) @Out (required=false)
  private Todo todo;

  @PersistenceContext (type=EXTENDED)
  private EntityManager em;
```

```
// Injected from pages.xml
Long id;

public String persist () {
  em.persist (todo);
  return "persisted";
}

@DataModel
private List <Todo> todos;

@Factory("todos")
public void findTodos () {
  todos = em.createQuery("select t from Todo t")
                                .getResultList();
}

public void setId (Long id) {
  this.id = id;

  if (id != null) {
    todo = (Todo) em.find(Todo.class, id);
  } else {
    todo = new Todo ();
  }
}

public Long getId () {
  return id;
}

public String delete () {
  em.remove( todo );
  return "removed";
}

public String update () {
  return "updated";
}

@Remove @Destroy
public void destroy() {}
```

```
}
```

First of all notice that this is a stateful session bean. Seam can use both stateful and stateless session beans, the two most common types of EJB3 beans.

The `@In` and `@Out` annotations define an attribute that is injected by Seam. The attribute is injected to this object or from this object to another via a Seam context variable named `todo`, a reference to the Seam registered name of our `Todo` class defined in `Todo.java`.

The `@PersistenceContext` annotation injects the EJB3 Entity manager, allowing this object to persist objects to the database. Because this is a stateful session bean and the `PersistenceContext` type is set to `EXTENDED`, the same Entity Manager instance is used until the Remove method of the session bean is called. The database to be used (a `persistence-unit`) is defined in the file `resources/META-INF/persistence.xml`

Note that this session bean has simultaneous access to context associated with web request (the form values of the `todo` object), and state held in transactional resources (the `EntityManager`). This is a break from traditional J2EE architectures, but Seam does not force you to work this way. You can use more traditional forms of application layering if you wish.

The `@DataModel` annotation initializes the `todos` property, which will be outjected or "exposed" to the view. The `@Factory` annotated method performs the work of generating the `todos` list, and is called by Seam if it attempts to access the exposed `DataModel` property and finds it to be null. Notice the absence of property access methods for the `todos` property. Seam takes care of this for you automatically.

Let's take a look at the JSF code that we use for displaying and editing the list of todos, to get an idea of how to use these interfaces in practice.

13.4. JSF Web Pages - todos.xhtml and edit.xhtml

Using the `DataModel` exposed property of the Session Bean it becomes trivial to produce a list of todos:

```
<h:form>

<h:dataTable value="#{todos}" var="todo">
  <h:column>
    <f:facet name="header">Title</f:facet>
    #{todo.title}
  </h:column>
  <h:column>
    <f:facet name="header">Description</f:facet>
    #{todo.description}
  </h:column>
  <h:column>
    <a href="edit.seam?tid=#{todo.id}">Edit</a>
  </h:column>
</h:dataTable>

<center>
  <h:commandButton action="create"
            value="Create New Todo" type="submit"/>
</center>

</h:form>
```

When the JSF variable resolver encounters `{#todos}` and requests `todos`, Seam finds that there is no "todos" component in the current scope, so it calls the @Factory("todos") method to make one. The todos object is then outjected once the factory method is done since it is annotated with the @DataModel annotation.

Constructing the view for the edit page is similarly straight forward:

```
<h:form id="edit">

<f:facet name="beforeInvalidField">
  <h:graphicImage styleClass="errorImg" value="error.png"/>
</f:facet>
<f:facet name="afterInvalidField">
  <s:message styleClass="errorMsg" />
</f:facet>
<f:facet name="aroundInvalidField">
  <s:div styleClass="error"/>
</f:facet>

<s:validateAll>

<table>

  <tr>
    <td>Title:</td>
    <td>
      <s:decorate>
        <h:inputText id="title" value="#{todo.title}" size="15"/>
      </s:decorate>
    </td>
  </tr>
  <tr>
    <td>Description:</td>
    <td>
      <s:decorate>
        <h:inputTextarea id="description" value="#{todo.description}"/>
      </s:decorate>
    </td>
  </tr>

</table>

</s:validateAll>

<h:commandButton type="submit" id="update" value="Update"
                 action="#{todoDao.update}"/>

<h:commandButton type="submit" id="delete" value="Delete"
                 action="#{todoDao.delete}"/>
</h:form>
```

Here we see the same factors in play. JSF validation code taking advantage of the validation constraints defined in our Entity Bean, and the use of the `todoDao` Session Bean's `update` and `delete` methods to update the database.

The call from `todos.xhtml`: `edit.seam?tid=#{todo.id}` causes Seam to create a `todoDao` and set it's `id` property to `tid`. Setting its `id` property causes the `todoDao` to retrieve the appropriate record from the database.

The functionality that allows the edit page to be called with a parameter in this way is implemented through `pages.xml`. Let's have a look at the `pages.xml` file and how it is used by Seam applications.

13.5. Xml Files

Seam drastically reduces the amount of xml coding that needs to be done. One file that is of interest is the `pages.xml`, packaged in the `app.war` file's `WEB-INF` directory. This file is available in the `resources/WEB-INF` directory in the source code bundle. The `pages.xml` file is used to define page descriptions including Seam page parameters (HTTP `GET` parameters), page actions, page navigation rules, error pages etc. Among other things it can be used in a Seam application to define exception handlers and redirections.

In the case of our sample application we are using it to define a Seam page parameter. The `pages.xml` in this example contains the following code:

```
<page view-id="/edit.xhtml">
   <param name="tid" value="#{todoDao.id}"
        converterId="javax.faces.Long"/>
</page>
```

This defines a parameter named `tid` for the `edit.xhtml` page. When the `edit.xhtml` page is loaded, the HTTP `GET` request parameter `tid` is converted to a `Long` value and assigned to the `id` property of the `todoDao` object. You can have as many page parameters as required to bind HTTP `GET` request parameters to the back-end components in your application.

13.6. Further Information

This completes our walkthrough of the sample Seam application. For further, detailed information on developing applications using the Seam framework, please refer to the *The Seam Reference Guide* [http://docs.jboss.com/seam/2.0.0.GA/reference/en/html_single/].

Using other Databases

In the previous chapters, we've been using the JBossAS default datasource in our applications. This datasource is configured to use the embedded Hypersonic database instance shipped by default with the distribution. This datasource is bound to the JNDI name `java:/DefaultDS` and its descriptor is named `hsqldb-ds.xml` under the deploy directory

Having a database included with JBossAS is very convenient for running the server and examples out-of-the-box. However, this database is not a production quality database and as such should not be used with enterprise-class deployments. As a consequence of this JBoss Support does not provide any official support for Hypersonic.

In this chapter we will explain in details how to configure and deploy a datasource to connect JBossAS to the most popular database servers available on the market today.

14.1. DataSource Configuration Files

Datasource configuration file names end with the suffix `-ds.xml` so that they will be recognized correctly by the JCA deployer. The `docs/example/jca` directory contains sample files for a wide selection of databases and it is a good idea to use one of these as a starting point. For a full description of the configuration format, the best place to look is the DTD file `docs/dtd/jboss-ds_1_5.dtd`. Additional documentation on the files and the JBoss JCA implementation can also be found in the JBoss Application Server Guide available at *http://labs.jboss.com/projects/docs/*.

Local transaction datasources are configured using the `local-tx-datasource` element and XA-compliant ones using `xa-tx-datasource`. The example file `generic-ds.xml` shows how to use both types and also some of the other elements that are available for things like connection pool configuration. Examples of both local and XA configurations are available for Oracle, DB2 and Informix.

If you look at the example files `firebird-ds.xml`, `facets-ds.xml` and `sap3-ds.xml`, you'll notice that they have a completely different format, with the root element being `connection-factories` rather than `datasources`. These use an alternative, more generic JCA configuration syntax used with a pre-packaged JCA resource adapter. The syntax is not specific to datasource configuration and is used, for example, in the `jms-ds.xml` file to configure the JMS resource adapter.

We would also highly recommend consulting the JCA wiki pages at http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossJCA

Next, we'll work through some step-by-step examples to illustrate what's involved setting up a datasource for a specific database.

14.2. Using MySQL as the Default DataSource

The MySQL® database has become the world's most popular open source database thanks to its consistent fast performance, high reliability and ease of use. This database server is used in millions of installations ranging from large corporations to specialized embedded applications across every continent of the world. . In this section, we'll be using the community version of their database server (GA 5.0.45) and the latest JDBC driver (GA 5.1.5) both available at *http://www.mysql.com*.

14.2.1. Installing the JDBC Driver and Deploying the datasource

To make the JDBC driver classes available to the JBoss Application Server, copy the archive `mysql-mysql-connector-java-5.1.5-bin.jar` from the Connector/J distribution to the `lib` directory in the `default` server configuration (assuming that is the server configuration you’re running).

Then create a text file in the deploy directory called mysql-ds.xml with the following datasource descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
   <local-tx-datasource>
   <jndi-name>DefaultDS</jndi-name>
   <connection-url>jdbc:mysql://localhost:3306/test</connection-url>
   <driver-class>com.mysql.jdbc.Driver</driver-class>
   <user-name>root</user-name>
   <password>jboss</password>
                                                          <valid-connection-checker-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker</valid-
connection-checker-class-name>
   <metadata>
   <type-mapping>mySQL</type-mapping>
   </metadata>
   </local-tx-datasource>
</datasources>
```

The datasource is pointing at the database called test provided by default with MySQL 5.x. Remember to update the connection url attributes as well as the combo username/password to match your environment setup.

14.2.2. Testing the MySQL DataSource

Using the test client described in *Section 14.6, “Creating a JDBC client”*, you may now verify the proper installation of your datasource.

14.3. Configuring a datasource for Oracle DB

Oracle is one of the main players in the commercial database field and most readers will probably have come across it at some point. You can download it freely for non-commercial purposes from *http://www.oracle.com/technology/products/database/xe/index.html*

In this section, we'll connect the server to Oracle Database 10g Express Edition using the latest JDBC driver (11g) available at *http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html*

14.3.1. Installing the JDBC Driver and Deploying the DataSource

To make the JDBC driver classes available to JBoss Application Server, copy the archive ojdbc5.jar to the lib directory in the default server configuration (assuming that is the server configuration you're running).

Then create a text file in the `deploy` directory called `oracle-ds.xml` with the following datasource descriptor :

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>DefaultDS</jndi-name>
    <connection-url>jdbc:oracle:thin:@localhost:1521:xe</connection-url>
    <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
    <user-name>SYSTEM</user-name>
    <password>jboss</password>
                                                        <valid-connection-checker-class-
name>org.jboss.resource.adapter.jdbc.vendor.OracleValidConnectionChecker</valid-
connection-checker-class-name>
    <metadata>
      <type-mapping>Oracle9i</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

The datasource is pointing at the database/SID called “xe” provided by default with Oracle XE.

Of course, you need to update the connection url attributes as well as the username/password combination to match your environment setup.

14.3.2. Testing the Oracle DataSource

Before you can verify the datasource configuration, Oracle XE should be reconfigured to avoid port conflict with JBossAS as by default they both start a web server on port 8080.

Open up an Oracle SQLcommand line and execute the following commands:

```
SQL> connect;
Enter user-name: SYSTEM
Enter password:
Connected.
SQL> begin
2  dbms_xdb.sethttpport('8090');
3  end;
4  /
PL/SQL procedure successfully completed.
SQL> select dbms_xdb.gethttpport from dual;
GETHTTPPORT
-----------
8090
```

The web server started by Oracle XE to provide http-based administration tools is now running on port 8090. Start the JBossAS server instance as you would normally do. You are now ready to use the test client described in Chapter 6.5 to verify the proper installation of your datasource.

14.4. Configuring a datasource for Microsoft SQL Server 200x

In this section, we'll connect the server to MS SQL Server 2000 using the latest JDBC driver (v1.2) available at *http://msdn2.microsoft.com/en-us/data/aa937724.aspx*.

14.4.1. Installing the JDBC Driver and Deploying the DataSource

To make the JDBC driver classes available to JBoss Application Server, copy the archive `sqljdbc.jar` from the `sqljdbc_1.2` distribution to the `lib` directory in the default server configuration (assuming that is the server configuration you're running).

Then create a text file in the `deploy` directory called `mssql-ds.xml` with the following datasource descriptor :

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
  <jndi-name>DefaultDS</jndi-name>
  <connection-url>jdbc:sqlserver://localhost:1433;DatabaseName=pubs</connection-url>
  <driver-class>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver-class>
  <user-name>sa</user-name>
  <password>jboss</password>
  <check-valid-connection-sql>SELECT 1 FROM sysobjects</check-valid-connection-sql>
  <metadata>
   <type-mapping>MS SQLSERVER2000</type-mapping>
  </metadata>
  </local-tx-datasource>
</datasources>
```

The datasource is pointing at a database "pubs" provided by default with MS SQL Server 2000.

Remember to update the connection url attributes as well as the username/password combination to match your environment setup.

14.4.1.1. Testing the datasource

Using the test client described in *Section 14.6, "Creating a JDBC client"*, you may now verify the proper installation of your datasource.

14.5. Configuring JBoss Messaging Persistence Manager

The persistence manager of JBoss Messaging uses the default datasource to create tables to store messages, transaction data and other indexes. Configuration of "persistence" is grouped in `xxx-persistence-service.xml` files. JBoss Application Server ships with a default `hsqldb-persistence-service.xml` file, which configures the Messaging server to use the Hypersonic database instance that ships by default with the JBoss Application Server.

You can view the `hsqldb-persistence-service.xml` file in configurations based on the *all* or *default* configurations:

```
<JBoss_Home>/server/all/deploy/messaging/hsqldb-persistence-service.xml  and

 <JBoss_Home>/server/default/deploy/messaging/hsqldb-persistence-service.xml
```

Warning

Please note that the Hypersonic database is not recommended for production environments due to its limited support for transaction isolation and its low reliability under high load

More information on configuring JBoss Messaging can be found in the *JBoss AS Configuration Guide* [http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Server_Configuration_Guide/beta500/html-single/index.html#d0e3471].

14.6. Creating a JDBC client

When testing a newly configured datasource we suggest using some very basic JDBC client code embedded in a JSP page. First of all, you should create an exploded WAR archive under the deploy directory which is simply a folder named "`jdbcclient.war`". In this folder, create a text document named client.jsp and paste the code below:

```
<%@page contentType="text/html"
 import="java.util.*,javax.naming.*,javax.sql.DataSource,java.sql.*"
 %>
 <%

  DataSource ds = null;
  Connection con = null;
  PreparedStatement pr = null;
  InitialContext ic;
  try {
  ic = new InitialContext();
  ds = (DataSource)ic.lookup( "java:/DefaultDS" );
  con = ds.getConnection();
  pr = con.prepareStatement("SELECT USERID, PASSWD FROM JMS_USERS");
  ResultSet rs = pr.executeQuery();
  while (rs.next()) {
  out.println("<br> " +rs.getString("USERID") + " | " +rs.getString("PASSWD"));
  }
  rs.close();
  pr.close();
  }catch(Exception e){
  out.println("Exception thrown " +e);
  }finally{
  if(con != null){
  con.close();
 }
} %>
```

Open up a web browser and hit the url: *http://localhost:8080/jdbcclient/client.jsp*. A list of users and password should show up as a result of the jdbc query:

```
dynsub | dynsub
guest | guest
j2ee | j2ee
john | needle
nobody | nobody
```

Appendix A.

Revision History

Revision 5.0.0	Jan 08 2007	SKittoli
Updated Content		
Revision 5.0.0	Apr 07 2007	SKittoli
Merge Content		
Revision 5.0.0.GA	Dec 08 2008	SStark
Complete the		

Appendix B. Further Information Sources

Developers wanting to get familiar with software development and implementation in JBoss Application Server can read: *JBoss: A Developer's Notebook*. (O'Reilly, 2005. Norman Richards, Sam Griffith).

For more comprehensive JBoss documentation covering advanced JBoss topics, refer to the manuals available online at *http://www.jboss.org/jbossas/docs*.

For general EJB instruction, with thorough JBoss coverage, see *Enterprise JavaBeans, 4th Edition*. (O'Reilly, 2004. Richard Monson-Haeful, Bill Burke, Sacha Labourey)

To learn more about Hibernate, see *Java Persistence With Hibernate*. (Manning, 2007. Christian Bauer, Gavin King)

For complete coverage of the JBoss Seam framework, we recommend *JBoss Seam: Simplicity And Power Beyond Java EE*. (Prentice Hall, 2007. Michael Yuan, Thomas Heute).