JBoss Enterprise Application Platform 6.2

Migration Guide

For Use with Red Hat JBoss Enterprise Application Platform 6

Edition 1

Last Updated: 2017-10-16

JBoss Enterprise Application Platform 6.2 Migration Guide

For Use with Red Hat JBoss Enterprise Application Platform 6
Edition 1

Nidhi Chaudhary

Lucas Costi

Russell Dickenson

Sande Gilda

Vikram Goyal

Eamon Logue

Darrin Mison

Scott Mumford

David Ryan

Misty Stanley-Jones

Keerat Verma

Tom Wells

Legal Notice

Abstract

This book is a guide to migrating your application from previous versions of Red Hat JBoss Enterprise Application Platform.

Table of Contents

CHAPTER 1. INTRODUCTION

1.1. ABOUT RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) is a middleware platform built on open standards and compliant with the Java Enterprise Edition 6 specification. It integrates JBoss Application Server 7 with high-availability clustering, messaging, distributed caching, and other technologies.

JBoss EAP 6 includes a new, modular structure that allows service enabling only when required, improving start-up speed.

As well, the Management Console and Management Command Line Interface make editing XML configuration files unnecessary and add the ability to script and automate tasks.

In addition, JBoss EAP 6 includes APIs and development frameworks for quickly developing secure and scalable Jave EE applications.

Report a bug

1.2. ABOUT THE MIGRATION GUIDE

JBoss EAP 6 is a fast, lightweight, powerful implementation of the Java Enterprise Edition 6 specification. The architecture is built on the Modular Service Container and enables services on-demand when your application requires them. Due to this new architecture, applications that run on JBoss EAP 5 may need modifications to run on JBoss EAP 6.

The purpose of this guide is to document the changes that are required to successfully run and deploy JBoss EAP 5.1 applications on JBoss EAP 6. It provides information on how to resolve deployment and runtime problems and to how prevent changes in application behavior. This is the first step in moving to the new platform. Once the application is successfully deployed and running, plans can be made to upgrade individual components to use the new functions and features of JBoss EAP 6.

Report a bug

CHAPTER 2. PREPARE FOR MIGRATION

2.1. PREPARE FOR MIGRATION

Because the application server is structured differently than in previous versions, you may want to do some research and planning before you attempt to migrate your application.

1. **Review What's New and Different in JBoss EAP 6**
   A number of things have changed in this release that may impact deployment of JBoss EAP 5 applications. These include changes to the file directory structure, scripts, deployment configuration, class loading and JNDI lookups. See Section 2.2, “Review What's New and Different in JBoss EAP 6” for details.

2. **Review the Get Started documentation**
   Be sure to review the chapter entitled *Get Started Developing Applications* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/. It contains important information about the following:

   - Java EE 6
   - The new modular class loading system
   - File structure changes
   - How to download and install JBoss EAP 6
   - How to download and install JBoss Developer Studio
   - How to configure Maven for your development environment
   - How to download and run the quickstart example applications that ship with the product.

3. **Learn How to Use JBoss EAP 6 Dependencies in your Maven Project**
   Be sure to review the chapter entitled *Maven Guide* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/. The *Manage Project Dependencies* section contains important information on how to configure your project to use JBoss EAP Bill of Material (BOM) artifacts.

4. **Analyze and Understand your Application**
   Each application is unique and you must thoroughly understand the components and architecture of the existing application before you attempt the migration.

IMPORTANT

Before making any modifications to your application, make sure to create a backup copy.

Report a bug

2.2. REVIEW WHAT'S NEW AND DIFFERENT IN JBOSS EAP 6

Introduction

The following is a list of notable differences in JBoss EAP 6 from the previous release.

Module based class loading

In JBoss EAP 5, the class loading architecture was hierarchical. In JBoss EAP 6, class loading is based on JBoss Modules. This offers true application isolation, hides server implementation classes, and only loads the classes your application needs. Class loading is concurrent for better performance. Applications written for JBoss EAP 5 must be modified to specify module dependencies and in some cases, repackage archives. For more information, refer to *Class Loading and Modules* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Domain Management

In JBoss EAP 6, the server can be run as a standalone server or in a managed domain. In a managed domain, you can configure entire groups of servers at once, keeping configurations synchronized across your entire network of servers. While this should not impact applications built for previous releases, this can simplify management of deployments to multiple servers. For more information, refer to *About Managed Domains* in the *Administration and Configuration Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Deployment Configuration

Standalone Servers and Managed Domains

JBoss EAP 5 used profile based deployment configuration. These profiles were located in the ***EAP_HOME*/server/** directory. Applications often contained multiple configuration files for security, database, resource adapter, and other configurations. In JBoss EAP 6, deployment configuration is done using one file. This file is used to configure all the services and subsystems used for the deployment. A standalone server is configured using the ***EAP_HOME*/standalone/configuration/standalone.xml** file. For servers running in a managed domain, the server is configured using the ***EAP_HOME*/domain/configuration/domain.xml** file. The information contained in the multiple JBoss EAP 5 configuration files must be migrated to the new single configuration file.

Ordering of deployments

JBoss EAP 6 uses fast, concurrent initialization for deployment resulting in improved performance and efficiency. In most cases, the application server is able to automatically determine dependencies in advance and choose the most efficient deployment strategy. However, JBoss EAP 5 applications that consist of multiple modules deployed as EARs and use legacy JNDI lookups instead of CDI injection or resource-ref entries may require configuration changes.

Directory Structure and Scripts

As previously mentioned, JBoss EAP 6 no longer uses profile based deployment configuration, so there is no ***EAP_HOME*/server/** directory. Configuration files for standalone servers are now located in the ***EAP_HOME*/standalone/configuration/** directory and deployments are located in the ***EAP_HOME*/standalone/deployments/** directory. For servers running in a managed domain, configuration files can be found in the ***EAP_HOME*/domain/configuration/** directory.

In JBoss EAP 5, the Linux script ***EAP_HOME*/bin/run.sh** or Windows script ***EAP_HOME*/bin/run.bat** was used to start the server. In JBoss EAP 6, the server start script is dependent on how you run your server. The Linux script ***EAP_HOME*/bin/standalone.sh** or Windows script ***EAP_HOME*/bin/standalone.bat** is used to start a standalone server. The Linux script ***EAP_HOME*/bin/domain.sh** or Windows script ***EAP_HOME*/bin/domain.bat** is used to start a managed domain.

JNDI Lookups

JBoss EAP 6 now uses standardized portable JNDI namespaces. Applications written for JBoss EAP 5 that use JNDI lookups must be changed to follow the new standardized JNDI namespace convention. For more information about JNDI naming syntax, see Section 3.1.8.2, “Portable EJB JNDI Names”.

For additional information, refer to *New and Changed Features in JBoss EAP 6* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Report a bug

2.3. REVIEW THE LIST OF DEPRECATED AND UNSUPPORTED FEATURES

Before you migrate your application, you should be aware that some features that were available in previous releases of JBoss EAP may be deprecated or no longer supported. For a comprehensive list, refer to the *Unsupported Features* section of the *Release Notes* for JBoss EAP 6 located on the Customer Portal at https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Report a bug

CHAPTER 3. MIGRATE YOUR APPLICATION

3.1. CHANGES REQUIRED BY MOST APPLICATIONS

3.1.1. Review Changes Required by Most Applications

Class loading and configuration changes in JBoss EAP 6 will impact almost every application. JBoss EAP 6 also uses new standard portable JNDI naming syntax. These changes will impact most applications, so it is suggested you review the following information first when you migrate your application.

1. Section 3.1.2.1, “Update the Application Due to Class Loading Changes”
2. Section 3.1.6.1, “Update the Application Due to Configuration Changes”
3. Section 3.1.8.1, “Update Application JNDI Namespace Names”

Report a bug

3.1.2. Class Loading Changes

3.1.2.1. Update the Application Due to Class Loading Changes

Modular class loading is a significant change in JBoss EAP 6 and will impact almost every application. Review the following information first when you migrate your application.

1. First, look at the packaging of your application and its dependencies. For more information, see: Section 3.1.2.3, “Update Application Dependencies Due to Class Loading Changes”
2. If your application does logging, you need to specify the correct module dependencies. For more information, see: Section 3.1.4.1, “Modify Logging Dependencies”
3. Due to the modular class loading changes, you may have to change the packaging structure of your EAR or WAR. For more information, see: Section 3.1.5.1, “Modify Packaging of EARs and WARs”

Report a bug

3.1.2.2. Understand Module Dependencies

Summary

A module is only able to access its own classes and the classes of any module on which it has an explicit or implicit dependency.

Procedure 3.1. Understand Module Dependencies

1. **Understand implicit dependencies**
   The deployers within the server implicitly automatically add some commonly used module dependencies, like the `javax.api` and `sun.jdk`. This makes the classes visible to the deployment at runtime and relieves the developer of the task of explicitly adding the dependencies. For details on how and when these implicit dependencies are added, refer to

Implicit Module Dependencies in the chapter entitled *Class Loading and Modules* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

2. **Understand explicit dependencies**
   For other classes, the modules must be specified explicitly or else the missing dependencies result in deployment or runtime errors. If a dependency is missing, you see **`ClassNotFoundExceptions`** or **`NoClassDefFoundErrors`** traces in the server log. If more than one module loads the same JAR or a module loads a class that extends a class loaded by a different module, you see **`ClassCastExceptions`** traces in the server log. To specify dependencies explicitly, modify the **`MANIFEST.MF`** or create a JBoss specific deployment descriptor file **`jboss-deployment-structure.xml`**. For more information on module dependencies, refer to *Overview of Class Loading and Modules* in the chapter entitled *Class Loading and Module* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Report a bug

3.1.2.3. Update Application Dependencies Due to Class Loading Changes

Summary

Class loading in JBoss EAP 6 is considerably different than in previous versions of JBoss EAP. Class loading is now based on the JBoss Modules project. Rather than a single, hierarchical class loader that loads all JARs into a flat class path, each library becomes a module that only links against the exact modules on which it depends. Deployments in JBoss EAP 6 are also modules and do not have access to classes that are defined in JARs in the application server unless an explicit dependency on those classes is defined. Some module dependencies defined by the application server are set up for you automatically. For instance, if you are deploying a Java EE application, a dependency on the Java EE API is added automatically, or implicitly. For the complete list of dependencies automatically added by the server, refer to *Implicit Module Dependencies* in the chapter entitled *Class Loading and Modules* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Tasks

When you migrate your application to JBoss EAP 6, you may need to perform one or more of the following tasks due to the modular class loading changes:

- Section 3.1.2.2, “Understand Module Dependencies”
- Section 4.1.3, “Use Tattletale to Find Application Dependencies”
- Section 3.1.3.1, “Create or Modify Files That Control Class Loading in JBoss EAP 6”
- Section 3.1.3.3, “Package Resources for the New Modular Class Loading System”

Report a bug

3.1.3. Configuration File Changes

3.1.3.1. Create or Modify Files That Control Class Loading in JBoss EAP 6

Summary

Due to the change in JBoss EAP 6 to use modular class loading, you may need to create or modify one or more files to add dependencies or to prevent automatic dependencies from loading. For more

information on class loading and class loading precedence, refer to the chapter entitled *Class Loading and Modules* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

The following files are used to control class loading in JBoss EAP 6.

jboss-web.xml

If you have defined a **<class-loading>** element in the **jboss-web.xml** file, you need to remove it. The behavior that this evoked in JBoss EAP 5 is now the default class loading behavior in JBoss EAP 6, so it is no longer necessary. If you do not remove this element, you see a ParseError and XMLStreamException in your server log.

This is an example of a **<class-loading>** element in the **jboss-web.xml** file that is commented out.

```
<!DOCTYPE jboss-web PUBLIC
    "-//JBoss//DTD Web Application 4.2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
<!--
    <class-loading java2ClassLoadingCompliance="false">
        <loader-repository>
            seam.jboss.org:loader=MyApplication
            <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
        </loader-repository>
    </class-loading>
 -->
 </jboss-web>
```

MANIFEST.MF

Manually edited

Depending on which components or modules your application uses, you may need to add one or more dependencies to this file. You can add them as either **Dependencies** or **Class-Path** entries.

The following is an example of **MANIFEST.MF** edited by a developer:

```
Manifest-Version: 1.0
Dependencies: org.jboss.logmanager
Class-Path: OrderManagerEJB.jar
```

If you modify this file, make sure to include a newline character at the end of the file.

Generated using Maven

If you use Maven, you need to modify your **pom.xml** file to generate the dependencies for the **MANIFEST.MF** file. If your application uses EJB 3.0, you may have a section in the **pom.xml** file that looks like the following:

```
<plugin>
```

```
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-ejb-plugin</artifactId>
    <configuration>
        <ejbVersion>3.0</ejbVersion>
    </configuration>
</plugin>
```

If the EJB 3.0 code uses **`org.apache.commons.log`**, you need that dependency in the **`MANIFEST.MF`** file. To generate that dependency, add the **`<plugin>`** element to the **`pom.xml`** file as follows:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-ejb-plugin</artifactId>
    <configuration>
        <ejbVersion>3.0</ejbVersion>
        <archive>
            <manifestFile>src/main/resources/META-
INF/MANIFEST.MF</manifestFile>
        </archive>
    </configuration>
</plugin>
```

In the above example, the **`src/main/resources/META-INF/MANIFEST.MF`** file only needs to contain the dependency entry:

```
Dependencies: org.apache.commons.logging
```

Maven will generate the complete **`MANIFEST.MF`** file:

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

jboss-deployment-structure.xml

This file is a JBoss specific deployment descriptor that can be used to control class loading in a fine grained manner. Like the **`MANIFEST.MF`**, this file can be used to add dependencies. It can also prevent automatic dependencies from being added, define additional modules, change an EAR deployment's isolated class loading behavior, and add additional resource roots to a module.

The following is an example of a **`jboss-deployment-structure.xml`** file that adds a dependency for JSF 1.2 module and prevents the automatic loading of the JSF 2.0 module.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
             <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
```

```
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
              <module name="com.sun.jsf-impl" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
              <module name="com.sun.jsf-impl" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

For additional information about this file, see: Section 3.1.3.2, “jboss-deployment-structure.xml”.

application.xml

In previous versions of JBoss EAP, you controlled the order of deployments within an EAR using the **`jboss-app.xml`** file. This is no longer the case. The Java EE6 spec provides the **`<initialize-in-order>`** element in the **`application.xml`** which allows control of the order in which the Java EE modules within an EAR are deployed.

In most cases you do not need to specify deployment order. If your application uses dependency injections and resource-refs to refer to components in external modules, in most cases the **`<initialize-in-order>`** element is not required because the application server is able to implicitly determine the correct and optimal way of ordering the components.

Let's assume you have an application that contains a **`myBeans.jar`** and a **`myApp.war`** within a **`myApp.ear`**. A servlet in the **`myApp.war@EJB`** injects a bean from **`myBeans.jar`**. In this case, the application server has the appropriate knowledge to make sure that the EJB component is available before the servlet is started and you do not have to use the **`<initialize-in-order>`** element.

However, if that servlet uses legacy JNDI lookup style remote references like the following to access the bean, you may need to specify module order.

```
init() {
  Context ctx = new InitialContext();
  ctx.lookup("TheBeanInMyBeansModule");
}
```

In this case, the server is not able to determine that the EJB component is in the **`myBeans.jar`** and you need to enforce that the components in the **`myBeans.jar`** are initialized and started before the components in **`myApp.war`**. To do this, you set the **`<initialize-in-order>`** element to **`true`** and specify the order of the **`myBeans.jar`** and **`myApp.war`** modules in the **`application.xml`** file.

The following is an example that uses the **`<initialize-in-order>`** element to control deployment order. The **`myBeans.jar`** is deployed before the **`myApp.war`** file.

```
<application xmlns="http://java.sun.com/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="6"
             xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
             http://java.sun.com/xml/ns/javaee/application_6.xsd">
    <application-name>myApp</application-name>
    <initialize-in-order>true</initialize-in-order>
    <module>
        <ejb>myBeans.jar</ejb>
```

```
    </module>
    <module>
        <web>
            <web-uri>myApp.war</web-uri>
            <context-root>myApp</context-root>
        </web>
    </module>
</application>
```

The schema for the **application.xml** file can be found here at http://java.sun.com/xml/ns/javaee/application_6.xsd.

NOTE

You should be aware that setting the **<initialize-in-order>** element to **true** slows down deployment. It is preferable to define proper dependencies using dependency injections or resource-refs because it allows the container more flexibility in optimizing deployments.

jboss-ejb3.xml

The **jboss-ejb3.xml** deployment descriptor replaces the **jboss.xml** deployment descriptor to override and add to the features provided by the Java Enterprise Edition (EE) defined **ejb-jar.xml** deployment descriptor. The new file is incompatible with **jboss.xml**, and the **jboss.xml** is now ignored in deployments.

login-config.xml

The **login-config.xml** file is no longer used for security configuration. Security is now configured in the **<security-domain>** element in the server configuration file. For a standalone server, this is the **standalone/configuration/standalone.xml** file. If you are running your server in a managed domain, this is the **domain/configuration/domain.xml** file.

Report a bug

3.1.3.2. jboss-deployment-structure.xml

jboss-deployment-structure.xml is a new optional deployment descriptor for JBoss EAP 6. This deployment descriptor provides control over class loading in the deployment.

The XML schema for this deployment descriptor is in ***EAP_HOME*/docs/schema/jboss-deployment-structure-1_2.xsd**

Report a bug

3.1.3.3. Package Resources for the New Modular Class Loading System

Summary

In previous versions of JBoss EAP, all resources inside the **WEB-INF/** directory were added to the WAR classpath. In JBoss EAP 6, web application artifacts are only loaded from the **WEB-INF/classes** and **WEB-INF/lib** directories. Failure to package application artifacts in the specified locations can result in **ClassNotFoundException**, **NoClassDefError**, or other runtime errors.

To resolve these class loading errors, you must modify the structure of your application archive or define a custom module.

Modify the Resource Packaging

To make the resources available only to the application, you must bundle the properties files, JARs, or other artifacts with the WAR by moving them to the **`WEB-INF/classes/`** or **`WEB-INF/lib/`** directory. This approach is described in more detail here: Section 3.1.3.4, “Change ResourceBundle Properties Location”

Create a Custom Module

If you want make custom resources available to all applications running on the JBoss EAP 6 server, you must create a custom module. This approach is described in more detail here: Section 3.1.3.5, “Create a Custom Module”

Report a bug

3.1.3.4. Change ResourceBundle Properties Location

Summary

In previous versions of JBoss EAP, the ***`EAP_HOME`*`/server/`*`SERVER_NAME`*`/conf/`** directory was in the classpath and available to the application. To make properties available to the classpath of the application in JBoss EAP 6, you must package them within your application.

Procedure 3.2. Change the ResourceBundle Properties Location

1. If you are deploying a WAR archive, you must package those properties in the WAR's **`WEB-INF/classes/`** folder.

2. If you want those properties accessible to all components in an EAR, then you must package them at the root of a JAR and then place the JAR in EAR's **`lib/`** folder.

Report a bug

3.1.3.5. Create a Custom Module

The following procedure describes how to create a custom module in order to make properties files and other resources available to all applications running on the JBoss EAP server.

Procedure 3.3. Create a Custom Module

1. Create and populate the **`module/`** directory structure.

   a. Create a directory structure under the ***`EAP_HOME`*`/module`** directory to contain the files and JARs. For example:

   ```
   $ cd EAP_HOME/modules/
   $ mkdir -p myorg-conf/main/properties
   ```

   b. Move the properties files to the ***`EAP_HOME`*`/modules/myorg-conf/main/properties/`** directory you created in the previous step.

c. Create a **module.xml** file in the ***EAP_HOME*/modules/myorg-conf/main/** directory containing the following XML:

```
<module xmlns="urn:jboss:module:1.1" name="myorg-conf">
    <resources>
        <resource-root path="properties"/>
    </resources>
</module>
```

2. Modify the **ee** subsystem in the server configuration file. You can use the JBoss CLI or you can manually edit the file.

- Follow these steps to modify the server configuration file using the JBoss CLI.

a. Start the server and connect to the Management CLI.

- For Linux, enter the following at the command line:

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

- For Windows, enter the following at a command line:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

You should see the following response:

```
Connected to standalone controller at localhost:9999
```

b. To create the **myorg-conf** <global-modules> element in the **ee** subsystem, type the following in the command line:

```
/subsystem=ee:write-attribute(name=global-modules, value=
[{"name"=>"myorg-conf","slot"=>"main"}])
```

You should see the following result:

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

- Follow these steps if you prefer to manually edit the server configuration file.

a. Stop the server and open the server configuration file in a text editor. If you are running a standalone server, this is the ***EAP_HOME*/standalone/configuration/standalone.xml** file, or the ***EAP_HOME*/domain/configuration/domain.xml** file if you are running a managed domain.

b. Find the **ee** subsystem and add the global module for **myorg-conf**. The following is an example of the **ee** subsystem element, modified to include the **myorg-conf** element:

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
    <global-modules>
        <module name="myorg-conf" slot="main" />
```

```
    </global-modules>
</subsystem>
```

3. Assuming you copied a file named **my.properties** into the correct module location, you are now able to load properties files using code similar to the following:

```
Thread.currentThread().getContextClassLoader().getResource("my.properties");
```

Report a bug

3.1.4. Logging Changes

3.1.4.1. Modify Logging Dependencies

Summary

JBoss LogManager supports front ends for all logging frameworks, so you can keep your current logging code or move to the new JBoss logging infrastructure. Regardless of your decision, because of the modular class loading changes, you probably need to modify your application to add the required dependencies.

Procedure 3.4. Update application logging code

1. Section 3.1.4.2, “Update Application Code for Third-party Logging Frameworks”
2. Section 3.1.4.3, “Modify Code to Use the New JBoss Logging Framework”

Report a bug

3.1.4.2. Update Application Code for Third-party Logging Frameworks

Summary

In JBoss EAP 6, logging dependencies for common third-party frameworks like Apache Commons Logging, Apache log4j, SLF4J, and Java Logging are added by default. However, if you are using log4j and you do not want to use the logging subsystem to configure your handlers (appenders), you need to exclude the JBoss EAP 6 log4j module.

Procedure 3.5. Configure JBoss EAP 6 to use a log4j.properties or log4j.xml file

1. Create a **jboss-deployment-structure.xml** with the following content:

```
<jboss-deployment-structure>
    <deployment>
        <!-- Exclusions allow you to prevent the server from automatically adding some dependencies -->
        <exclusions>
            <module name="org.apache.log4j" />
        </exclusions>
    </deployment>
</jboss-deployment-structure>
```

2. Place the **jboss-deployment-structure.xml** file in either the **META-INF/** directory or the **WEB-INF/** directory if you are deploying a WAR, or in the **META-INF/** directory if you are deploying an EAR.

3. Include the **log4j.properties** or **log4j.xml** file in the **lib/** directory of your EAR, or the **WEB-INF/classes/** directory of your WAR deployment.

4. Start the JBoss Enterprise Application Platform 6 server with the following runtime argument to prevent a **ClassCastException** from appearing in the console when you deploy the application:

```
-Dorg.jboss.as.logging.per-deployment=false
```

5. Deploy your application.

NOTE

If you choose to use a log4j configuration file, you will no longer be able to change the log4j logging configuration at runtime.

Report a bug

3.1.4.3. Modify Code to Use the New JBoss Logging Framework

Summary

To use the new framework, change your imports and code as follows:

Procedure 3.6. Modify Code and Dependencies to Use the JBoss Logging Framework

1. **Change your imports and logging code**
   The following is an example of code that uses the new JBoss Logging framework:

```
import org.jboss.logging.Level;
import org.jboss.logging.Logger;

private static final Logger logger =
Logger.getLogger(MyClass.class.toString());

if(logger.isTraceEnabled()) {
    logger.tracef("Starting...", subsystem);
}
```

2. **Add the logging dependency**
   The JAR containing the JBoss Logging classes is located in the module named **org.jboss.logging**. Your **MANIFEST-MF** file should look like this:

```
Manifest-Version: 1.0
Dependencies: org.jboss.logging
```

   For more information on how to find the module dependency, please see Section 3.1.2.3, “Update Application Dependencies Due to Class Loading Changes” and Section 4.2.1, “Debug and Resolve Migration Issues”.

Report a bug

3.1.5. Application Packaging Changes

3.1.5.1. Modify Packaging of EARs and WARs

Summary

When you migrate your application, you may have to change the packaging structure of your EAR or WAR due to the change to modular class loading. Module dependencies are loaded in this specific order:

1. System dependencies
2. User dependencies
3. Local resources
4. Inter-deployment dependencies

Procedure 3.7. Modify archive packaging

1. **Package a WAR**
   A WAR is a single module and all classes in the WAR are loaded with the same class loader. This means classes packaged in the **`WEB-INF/lib/`** directory are treated the same as classes in the **`WEB-INF/classes`** directory.

2. **Package an EAR**
   An EAR consists of multiple modules. The **`EAR/lib/`** directory is a single module and every WAR or EJB jar subdeployment within the EAR is a separate module. Classes do not have access to classes in other modules within the EAR unless explicit dependencies have been defined. Subdeployments always have an automatic dependency on the parent module which gives them access to classes in the **`EAR/lib/`** directory. However, subdeployments do not always have an automatic dependency to allow them to access each other. This behavior is controlled by setting the **`<ear-subdeployments-isolated>`** element in the **`ee`** subsystem configuration as follows:

   ```
   <subsystem xmlns="urn:jboss:domain:ee:1.0" >
     <ear-subdeployments-isolated>false</ear-subdeployments-isolated>
   </subsystem>
   ```

   By default this is set to false which allows the subdeployments to see classes belonging to other subdeployments within the EAR.

   For more information on class loading, refer to the chapter entitled *Class Loading and Modules* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Report a bug

3.1.6. Datasource and Resource Adapter Configuration Changes

3.1.6.1. Update the Application Due to Configuration Changes

In JBoss EAP 5, services and subsystems were configured in many different files. In JBoss EAP 6, configuration is now done mainly in one file. If your application uses any of the following resources or services, configuration changes may be needed.

1. If your application uses a datasource, see: Section 3.1.6.2, “Update the DataSource Configuration”.
2. If your application uses JPA and currently bundles the Hibernate JARs, see the following for your migration options: Section 3.1.6.4, “Configure the Datasource for Hibernate or JPA”.
3. If your application uses a resource adapter, see: Section 3.1.6.5, “Update the Resource Adapter Configuration”.
4. Review the following for information on how to configure changes for basic security: Section 3.1.7.1, “Configure Application Security Changes”.

Report a bug

3.1.6.2. Update the DataSource Configuration

Summary

In previous versions of JBoss EAP, the JCA datasource configuration was defined in a file with a suffix of **`*-ds.xml`**. This file was then deployed in the server's **`deploy/`** directory or packaged with the application. The JDBC driver was copied to the **`server/lib/`** directory or packaged in the application's **`WEB-INF/lib/`** directory. While this method of configuring a datasource is still supported for development, it is not recommended for production because it is not supported by the JBoss administrative and management tools.

In JBoss EAP 6, the datasource is configured in the server configuration file. If the JBoss EAP 6 instance is running in a managed domain, the datasource is configured in the **`domain/configuration/domain.xml`** file. If the JBoss EAP 6 instance is running as a standalone server, the datasource is configured in the **`standalone/configuration/standalone.xml file`**. Datasources configured this way can be managed and controlled using the JBoss management interfaces, including the Web Management Console and command line interface (CLI). These tools make it easy to manage deployments and configure multiple servers running in a managed domain.

The following section describes how to modify your datasource configuration so that it can be managed and supported by the available management tools.

Migrate to a Manageable Datasource Configuration for JBoss EAP 6

A JDBC 4.0 compliant driver can be installed as a deployment or as a core module. A driver that is JDBC 4.0 compliant contains a **`META-INF/services/java.sql.Driver`** file that specifies the driver class name. A driver that is not JDBC 4.0 compliant requires additional steps. For details on how to make a driver JDBC 4.0 compliant and how update your current datasource configuration to one that is manageable by the Web Management Console and CLI, see Section 3.1.6.3, “Install and Configure the JDBC Driver”.

If your application uses Hibernate or JPA, it may require additional changes. See Section 3.1.6.4, “Configure the Datasource for Hibernate or JPA” for more information.

Use the IronJacamar Migration Tool to Convert Configuration Data

You can use the IronJacamar tool to migrate datasource and resource adapter configurations. This tool converts the **`*-ds.xml`** style configuration files into the format expected by JBoss EAP 6. For more information, see: Section 4.1.6, “Use the IronJacamar Tool to Migrate Datasource and Resource Adapter Configurations”.

Report a bug

3.1.6.3. Install and Configure the JDBC Driver

Summary

The JDBC driver can be installed into the container in one of the following two ways:

- As a deployment
- As a core module

The pros and cons of each approach are noted below.

In JBoss EAP 6, the datasource is configured in the server configuration file. If the JBoss EAP 6 instance is running in a managed domain, the datasource is configured in the **`domain/configuration/domain.xml`** file. If the JBoss EAP 6 instance is running as a standalone server, the datasource is configured in the **`standalone/configuration/standalone.xml`** file. Schema reference information, which is the same for both modes, can be found in the **`docs/schema/`** directory of the JBoss EAP 6 install. For purposes of this discussion, assume the server is running as standalone server and the datasource is configured in the **`standalone.xml`** file.

Procedure 3.8. Install and Configure the JDBC Driver

1. **Install the JDBC Driver**

   a. **Install the JDBC Driver as a deployment**
   This is the recommended way to install the driver. When the JDBC driver is installed as a deployment, it is deployed as a regular JAR. If the JBoss EAP 6 instance is running as a standalone server, copy the JDBC 4.0 compliant JAR into the ***`EAP_HOME`*`/standalone/deployments/`** directory. For a managed domain, you must use the Management Console or Managment CLI to deploy the JAR to the server groups.

   The following is an example of a MySQL JDBC driver installed as a deployment to a standalone server:

   ```
   $cp mysql-connector-java-5.1.15.jar
   EAP_HOME/standalone/deployments/
   ```

   Any JDBC 4.0 compliant driver is automatically recognized and installed into the system by name and version. A JDBC 4.0 compliant JAR contains a text file named **`META-INF/services/java.sql.Driver`** which specifies the driver class name(s). If the driver is not JDBC 4.0 compliant, it can be made deployable in one of the following ways:

   - Create and add a **`java.sql.Driver`** file to the JAR under the **`META-INF/services/`** path. This file should contain the driver class name, for example:

     ```
     com.mysql.jdbc.Driver
     ```

   - Create a **`java.sql.Driver`** file in the deployment directory. For a JBoss EAP 6 instance running as a standalone server, the file should be placed here: ***`EAP_HOME`*`/standalone/deployments/META-INF/services/java.sql.Driver`**. If the server is in a managed domain, you must use the Management Console or Managment CLI to deploy the file.

The pros of this approach are:

- This is the easiest method because there is no need to define a module.
- When the server is running in a managed domain, deployments that use this approach are automatically propagated to all servers in the domain. This means the administrator does not need to distribute the driver JAR manually.

The cons of this approach are:

- If the JDBC driver consists of more than one JAR, for example the driver JAR plus a dependent license JAR or localization JAR, you can not install the driver as a deployment. You must install the JDBC driver as a core module.
- If the driver is not JDBC 4.0 compliant, a file must be created containing the driver class name(s) and must be imported into the JAR or overlayed in the **`deployments/`** directory.

b. **Install the JDBC Driver as a core module**
To install a JDBC driver as a core module, you must create a file path structure under the ***`EAP_HOME`*`/modules/`** directory. This structure contains the JDBC driver JAR, any additional vendor license or localization JARs, and a **`module.xml`** file to define the module.

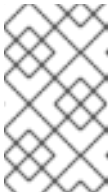
- **Install the MySQL JDBC Driver as a core module**

  i. Create the directory structure ***`EAP_HOME`*`/modules/com/mysql/main/`**

  ii. In the **`main/`** subdirectory, create a **`module.xml`** file containing the following module definition for the MySQL JDBC driver:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
<resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
</resources>
<dependencies>
    <module name="javax.api"/>
</dependencies>
</module>
```

The module name, "com.mysql", matches the directory structure for this module. The **`<dependencies>`** element is used to specify this module's dependencies on other modules. In this case, as is the case with all JDBC datasources, it is dependent on the Java JDBC APIs which are defined in another module named **`javax.api`**. That module is located under the **`modules/system/layers/base/javax/api/main/`** directory.



NOTE

Make sure you do NOT have a space at the beginning of **`module.xml`** file or you will get a "New missing/unsatisfied dependencies" error for this driver.

iii. Copy the MySQL JDBC driver JAR into the ***EAP_HOME*/modules/com/mysql/main/** directory:

```
$ cp mysql-connector-java-5.1.15.jar
EAP_HOME/modules/com/mysql/main/
```

- **Install the IBM DB2 JDBC driver and license JAR as a core module**
  This example is provided to only demonstrate how to deploy drivers that require JARs in addition to the JDBC Driver JAR.

  i. Create the directory structure ***EAP_HOME*/modules/com/ibm/db2/main/**.

  ii. In the **main/** subdirectory, create a **module.xml** file containing the following module definition for the IBM DB2 JDBC driver and license:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm.db2">
  <resources>
    <resource-root path="db2jcc.jar"/>
    <resource-root path="db2jcc_license_cisuz.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

NOTE

Make sure you do NOT have a space at the beginning of **module.xml** file or you will get a "New missing/unsatisfied dependencies" error for this driver.

  iii. Copy the JDBC driver and license JAR to the ***EAP_HOME*/modules/com/ibm/db2/main/** directory.

```
$ cp db2jcc.jar EAP_HOME/modules/com/ibm/db2/main/
$ cp db2jcc_license_cisuz.jar
EAP_HOME/modules/com/ibm/db2/main/
```

The pros of this approach are:

- This is the only approach that works when the JDBC driver consists of more than one JAR.
- With this approach, drivers that are not JDBC 4.0 compliant can be installed without modifying the driver JAR or creating a file overlay.

The cons of this approach are:

- It is more difficult to set up a module.
- The module must be manually copied to every server running in a managed domain.

2. **Configure the datasource**

   a. **Add the database driver**
   Add the **`<driver>`** element to the **`<drivers>`** element of the same file. Again, this contains some of the same datasource information that was previously defined in the **`*-ds.xml`** file.

   First determine if the driver JAR is JDBC 4.0 compliant. A JAR that is JDBC 4.0 compliant contains a **`META-INF/services/java.sql.Driver`** file that specifies the driver class name. The server uses this file to find the name of the driver class(es) in the JAR. A driver that is JDBC 4.0 compliant does not require a **`<driver-class>`** element since it is already specified in the JAR. This is an example of the driver element for a JDBC 4.0 compliant MySQL driver:

   ```
   <driver name="mysql-connector-java-5.1.15.jar"
   module="com.mysql"/>
   ```

   A driver that is not JDBC 4.0 compliant requires a **`<driver-class>`** attribute to identify the driver class since there is no **`META-INF/services/java.sql.Driver`** file that specifies the driver class name. This is an example of the driver element for driver that is not JDBC 4.0 compliant:

   ```
   <driver name="mysql-connector-java-5.1.15.jar"
   module="com.mysql">
   <driver-class>com.mysql.jdbc.Driver</driver-class></driver>
   ```

   b. **Create the datasource**
   Create a **`<datasource>`** element in the **`<datasources>`** section of the **`standalone.xml`** file. This file contains much of the same datasource information that was previously defined in the **`*-ds.xml`** file.

   IMPORTANT

   You must stop the server before editing the server configuration file for your change to be persisted on server restart.

   The following is an example of a MySQL datasource element in the **`standalone.xml`** file:

   ```
   <datasource jndi-name="java:/YourDatasourceName" pool-
   name="YourDatasourceName">
     <connection-
   url>jdbc:mysql://localhost:3306/YourApplicationURL</connection-
   url>
     <driver>mysql-connector-java-5.1.15.jar</driver>
     <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
   isolation>
     <pool>
       <min-pool-size>100</min-pool-size>
       <max-pool-size>200</max-pool-size>
     </pool>
     <security>
       <user-name>USERID</user-name>
       <password>PASSWORD</password>
     </security>
     <statement>
   ```

```
    <prepared-statement-cache-size>100</prepared-statement-cache-
size>
    <share-prepared-statements/>
  </statement>
</datasource>
```

Report a bug

3.1.6.4. Configure the Datasource for Hibernate or JPA

If your application uses JPA and currently bundles the Hibernate JARs, you may want to use the Hibernate that is included with JBoss EAP 6. To use this version of Hibernate, you must remove the old Hibernate bundle from your application.

Procedure 3.9. Remove the Hibernate bundle

1. Remove the Hibernate JARs from your application library folders.
2. Remove or comment out the **`<hibernate.transaction.manager_lookup_class>`** element in your **`persistence.xml`** file as this element is not needed.

Report a bug

3.1.6.5. Update the Resource Adapter Configuration

Summary

In previous versions of the application server, the resource adapter configuration was defined in a file with a suffix of **`*-ds.xml`**. In JBoss EAP 6, a resource adapter is configured in the server configuration file. If you are running in a managed domain, the configuration file is the ***`EAP_HOME`*`/domain/configuration/domain.xml`** file. If you are running as a standalone server, configure the resource adapter in the ***`EAP_HOME`*`/standalone/configuration/standalone.xml`** file. Schema reference information, which is the same for both modes, can be found under *Schemas* on the IronJacamar web site here: http://www.ironjacamar.org/documentation.html.

IMPORTANT

You must stop the server before editing the server configuration file for your change to be persisted on server restart.

Define the resource adapter

The resource adapter descriptor information is defined under the following subsystem element in the server configuration file:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1"/>
```

You will use some of the same information that was previously defined in the resource adapter **`*-ds.xml`** file.

The following is an example of a resource adapter element in the server configuration file:

```
<resource-adapters>
```

```
  <resource-adapter>
    <archive>multiple-full.rar</archive>
    <config-property name="Name">ResourceAdapterValue</config-property>
    <transaction-support>NoTransaction</transaction-support>
    <connection-definitions>
      <connection-definition
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory1"
      enabled="true" jndi-name="java:/eis/MultipleConnectionFactory1"
      pool-name="MultipleConnectionFactory1">
    <config-property name="Name">MultipleConnectionFactory1Value</config-
property>
      </connection-definition>
      <connection-definition
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory2"
      enabled="true" jndi-name="java:/eis/MultipleConnectionFactory2"
      pool-name="MultipleConnectionFactory2">
    <config-property name="Name">MultipleConnectionFactory2Value</config-
property>
      </connection-definition>
    </connection-definitions>
    <admin-objects>
      <admin-object
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject1
Impl"
      jndi-name="java:/eis/MultipleAdminObject1">
    <config-property name="Name">MultipleAdminObject1Value</config-
property>
      </admin-object>
      <admin-object class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject2
Impl"
      jndi-name="java:/eis/MultipleAdminObject2">
    <config-property name="Name">MultipleAdminObject2Value</config-
property>
      </admin-object>
      </admin-objects>
  </resource-adapter>
</resource-adapters>
```

Report a bug

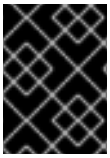
3.1.7. Security Changes

3.1.7.1. Configure Application Security Changes

Configure security for basic authentication

In previous versions of JBoss EAP, properties files placed in the ***EAP_HOME*/server/*SERVER_NAME*/conf/** directory were on classpath and could be easily found by the **UsersRolesLoginModule**. In JBoss EAP 6, the directory structure has changed. Properties files

must be packaged within the application to make them available in the classpath.



IMPORTANT

You must stop the server before editing the server configuration file for your change to be persisted on server restart.

To configure security for basic authentication, add a new security domain under **`security-domains`** to the **`standalone/configuration/standalone.xml`** or the **`domain/configuration/domain.xml`** server configuration file:

```
<security-domain name="example">
    <authentication>
        <login-module code="UsersRoles" flag="required">
            <module-option name="usersProperties"
                    value="${jboss.server.config.dir}/example-
users.properties"/>
            <module-option name="rolesProperties"
                    value="${jboss.server.config.dir}/example-
roles.properties"/>
        </login-module>
    </authentication>
</security-domain>
```

If the JBoss EAP 6 instance is running as a standalone server, **`${jboss.server.config.dir}`** refers to the ***`EAP_HOME`*`/standalone/configuration/`** directory. If the instance is running in a managed domain, **`${jboss.server.config.dir}`** refers to the ***`EAP_HOME`*`/domain/configuration/`** directory.

Modify security domain names

In JBoss EAP 6, security domains no longer use the prefix **`java:/jaas/`** in their names.

- For Web applications, you must remove this prefix from the security domain configurations in the **`jboss-web.xml`**.
- For Enterprise applications, you must remove this prefix from the security domain configurations in the **`jboss-ejb3.xml`** file. This file has replaced the **`jboss.xml`** in JBoss EAP 6.

Report a bug

3.1.8. JNDI Changes

3.1.8.1. Update Application JNDI Namespace Names

Summary

EJB 3.1 introduced a standardized global JNDI namespace and a series of related namespaces that map to the various scopes of a Java EE application. Portable EJB names only get bound to three of them: **`java:global`**, **`java:module`**, and **`java:app`**. Applications with EJBs that use JNDI must be changed to follow the new standardized JNDI namespace convention.

Procedure 3.10. Modify JNDI lookups

1. Learn more about Section 3.1.8.2, “Portable EJB JNDI Names”
2. Section 3.1.8.3, “Review the JNDI Namespace Rules”
3. Section 3.1.8.4, “Modify the Application to Follow the New JNDI Namespace Rules”

Example JNDI Mappings

Examples of JNDI namespaces in previous releases and how they are specified in JBoss EAP 6 can be found here: Section 3.1.8.5, “Examples of JNDI Namespaces in Previous Releases and How They are Specified in JBoss EAP 6”

Report a bug

3.1.8.2. Portable EJB JNDI Names

Summary

The Java EE 6 specification defines four logical namespaces, each with its own scope, but portable EJB names only get bound to three of them. The following table details when and how to use each namespace.

Table 3.1. Portable JNDI Namespaces

JNDI Namespace	Description
java:global	Names in this namespace are shared by all applications deployed in an application server instance. Use names in this namespace to find EJBs external archives deployed to the same server. The following is an example of a java:global namespace: **`java:global/jboss-seam-booking/jboss-seam-booking-jar/HotelBookingAction`**
java:module	Names in this namespace are shared by all components in a module, for example, all enterprise beans in a single EJB module or all components in a web module. The following is an example of a java:module namespace: **`java:module/HotelBookingAction!org.jboss.seam.example.booking.HotelBooking`**
java:app	Names in this namespace are shared by all components in all modules in a single application. For example, a WAR and an EJB jar file in the same EAR file would have access to resources in the java:app namespace. The following is an example of a java:app namespace: **`java:app/jboss-seam-booking-jar/HotelBookingAction`**

You can find more information about JNDI naming contexts in section EE.5.2.2, "Application Component Environment Namespaces" in the "JSR 316: JavaTM Platform, Enterprise Edition (Java EE) Specification, v6". You can download the specification from here: http://jcp.org/en/jsr/detail?id=316

Report a bug

3.1.8.3. Review the JNDI Namespace Rules

Summary

JBoss EAP 6 has improved upon JNDI namespace names, not only to provide predictable and consistent rules for every name bound in the application server, but also to prevent future compatibility issues. This means you might run into issues with the current namespaces in your application if they don't follow the new rules.

Namespaces should follow these rules:

1. Unqualified relative names like **`DefaultDS`** or **`jdbc/DefaultDS`** should be qualified relative to **`java:comp/env`**, **`java:module/env`**, or **`java:jboss/env`**, depending on the context.
2. Unqualified **`absolute`** names like **`/jdbc/DefaultDS`** should be qualified relative to a **`java:jboss/root`** name.
3. Qualified **`absolute`** names like **`java:/jdbc/DefaultDS`** should be qualified the same way as Unqualified **`absolute`** names above.
4. The special **`java:jboss`** namespace is shared across the entire AS server instance.
5. Any **`relative`** name with a **`java:`** prefix must be in one of the five namespaces: **`comp`**, **`module`**, **`app`**, **`global`**, or the proprietary **`jboss`**. Any name starting with **`java:xxx`** where *xxx* does not match any of the above five would result in an invalid name error.

Report a bug

3.1.8.4. Modify the Application to Follow the New JNDI Namespace Rules

- Here is an example of a JNDI lookup in JBoss EAP 5.1. This code is usually found in an initialization method.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("OrderManagerApp/ProductManagerBean/local");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

  Note the lookup name is **`OrderManagerApp/ProductManagerBean/local`**.

- The following is an example of how the same lookup would be coded in JBoss EAP 6 using CDI.

```
@EJB(lookup="java:app/OrderManagerEJB/ProductManagerBean!services.ej
b.ProductManager")
private ProductManager productManager;
```

  The lookup values are now defined as member variables and use the new portable **`java:app`** JNDI namespace name **`java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManage r`**.

- If you prefer not to use CDI, you can continue to create the new InitialContext as above and modify the lookup to use the new JNDI namespace name.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("java:app/OrderManagerEJB/ProductManagerBean!services
.ejb.ProductManager");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

Report a bug

3.1.8.5. Examples of JNDI Namespaces in Previous Releases and How They are Specified in JBoss EAP 6

Table 3.2. JNDI Namespace Mapping Table

Namespace in JBoss EAP 5.x	Namespace in JBoss EAP 6	Additional Comments
OrderManagerApp/ProductManagerBean/local	java:module/ProductManagerBean!services.ejb.ProductManager	Java EE 6 standard binding. Scoped to the current module and only accessible within the same module.
OrderManagerApp/ProductManagerBean/local	java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE 6 standard binding. Scoped to the current application and only accessible within the same application.
OrderManagerApp/ProductManagerBean/local	java:global/OrderManagerApp/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE 6 standard binding. Scoped to the application server and globally accessible.
java:comp/UserTransaction	java:comp/UserTransaction	Namespace is scoped to the current component. Not accessible for threads that are not Java EE 6, for example, threads created directly by your application.
java:comp/UserTransaction	java:jboss/UserTransaction	Globally accessible. Use this if java:comp/UserTransaction is not available.
java:/TransactionManager	java:jboss/TransactionManager	

Namespace in JBoss EAP 5.x	Namespace in JBoss EAP 6	Additional Comments
java:/TransactionSynchronization Registry	java:jboss/TransactionSynchroniz ationRegistry	

Report a bug

3.2. CHANGES DEPENDENT ON YOUR APPLICATION ARCHITECTURE AND COMPONENTS

3.2.1. Review Changes Dependent on Your Application Architecture and Components

If your application uses any of the following technologies or components, you may need to make modifications to your application when you migrate to JBoss EAP 6.

Hibernate and JPA

If your application uses Hibernate or JPA, your application may need some modifications. For more information, see: Section 3.2.2.1, “Update Applications That Use Hibernate and/or JPA”.

REST

If your application uses JAX-RS, you should be aware that JBoss EAP 6 automatically sets up RESTEasy, so you no longer need to configure it yourself. For more information, see: Section 3.2.5.1, “Configure JAX-RS and RESTEasy Changes”

LDAP

The LDAP security realm is configured differently in JBoss EAP 6. If your application uses LDAP, refer to the following topic for more information: Section 3.2.6.1, “Configure LDAP Security Realm Changes”.

Messaging

JBoss Messaging is no longer included in JBoss EAP 6. If your application uses JBoss Messaging as the messaging provider, you need to replace the JBoss Messaging code with HornetQ. The following topic describes what you need to do: Section 3.2.7.3, “Migrate Your Application to Use HornetQ as the JMS Provider”.

Clustering

The way you enable clustering has changed in JBoss EAP 6. For details, see: Section 3.2.8.1, “Make Changes to Your Application for Clustering”.

Service-style deployment

Although JBoss EAP 6 no longer uses service-style descriptors, the container supports these service-style deployments without change where possible. For deployment information, see: Section 3.2.9.1, “Update Applications That Use Service-style Deployments”

Remote invocation

If your application makes remote invocations, you can still use JNDI to lookup a proxy for your bean and invoke on that returned proxy. For more information about required syntax and namespaces

changes, see: Section 3.2.10.1, “Migrate JBoss EAP 5 Deployed Applications That Make Remote Invocations to JBoss EAP 6”.

Seam 2.2

If your application uses Seam 2.2, refer to the following topic for changes you need to make: Section 3.2.13.1, “Migrate Seam 2.2 Archives to JBoss EAP 6”.

Spring

If your application uses Spring, see: Section 3.2.14.1, “Migrate Spring Applications”.

Other changes that may impact your migration

For additional changes in JBoss EAP 6 that may impact your application, see: Section 3.2.15.1, “Become Familiar with Other Changes That May Affect Your Migration”.

Report a bug

3.2.2. Hibernate and JPA Changes

3.2.2.1. Update Applications That Use Hibernate and/or JPA

Summary

If your application uses Hibernate or JPA, read through the following sections and make any changes necessary to migrate to JBoss EAP 6.

- Section 3.2.2.2, “Configure Changes for Applications That Use Hibernate and JPA”
- Section 3.2.2.4, “Update Your Hibernate 3 Application to Use Hibernate 4”
- Section 3.2.2.9, “Update Your Application to Conform to the JPA 2.0 Specification”
- Section 3.2.2.10, “Replace JPA/Hibernate Second Level Cache with Infinispan”
- Section 3.2.2.12, “Migrate to Hibernate Validator 4”

Report a bug

3.2.2.2. Configure Changes for Applications That Use Hibernate and JPA

Summary

If your application contains a **`persistence.xml`** file or the code uses the annotations **`@PersistenceContext`** or **`@PersistenceUnit`**, JBoss EAP 6 detects this during deployment and assumes the application uses JPA. It implicitly adds Hibernate 4 plus a few other dependencies to your application classpath.

If your application currently uses Hibernate 3 libraries, in most cases you will be able to switch to using Hibernate 4 and run successfully. However, if you see **`ClassNotFoundExceptions`** when you deploy your application, you can try to resolve them using one of the following approaches.

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

Procedure 3.11. Configure the Application

1. **Copy the required Hibernate 3 JARs to your application library.**
   You may be able to resolve the issue by copying the specific Hibernate 3 JARs that contain the missing classes into the application's **`lib/`** directory or by adding them to the classpath using some other method. In some cases this may result in **`ClassCastExceptions`** or other class loading issues due to the mixed use of the Hibernate versions. If that happens, you need to use the next approach.

2. **Instruct the server to use only the Hibernate 3 libraries.**
   JBoss EAP 6 allows you to package Hibernate 3.5 (or greater) persistence provider jars with the application. To direct the server to use only the Hibernate 3 libraries and to exclude the Hibernate 4 libraries, you need to set the **`jboss.as.jpa.providerModule`** to **`hibernate3-bundled`** in the **`persistence.xml`** as follows:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <persistence xmlns="http://java.sun.com/xml/ns/persistence"
   version="1.0">
       <persistence-unit name="plannerdatasource_pu">
           <description>Hibernate 3 Persistence Unit.</description>
           <jta-data-source>java:jboss/datasources/PlannerDS</jta-data-
   source>
           <properties>
               <property name="hibernate.show_sql" value="false" />
               <property name="jboss.as.jpa.providerModule"
   value="hibernate3-bundled" />
           </properties>
       </persistence-unit>
   </persistence>
   ```

   The Java Persistence API (JPA) deployer will detect the presence of a persistence provider in the application and use the Hibernate 3 libraries. For more information on the JPA persistence properties, see Section 3.2.2.3, “Persistence Unit Properties”.

3. **Disable Hibernate second-level cache**
   Second-level cache for Hibernate 3 does not exhibit the same behavior with JBoss EAP 6 as it did in previous releases. If you are using Hibernate second-level cache with your application, you must disable it until you upgrade to Hibernate 4. To disable second-level cache, set the **`<hibernate.cache.use_second_level_cache>`** to **`false`** in the **`persistence.xml`** file.

Report a bug

3.2.2.3. Persistence Unit Properties

Hibernate 4.x configuration properties

JBoss EAP 6 automatically sets the following Hibernate 4.x configuration properties:

Table 3.3. Hibernate Persistence Unit Properties

Property Name	Default Value	Purpose
hibernate.id.new_generator_mappings	true	This setting is relevant if you use **@GeneratedValue(AUTO)** to generate unique index key values for new entities. New applications should keep the default value of **true**. Existing applications that used Hibernate 3.3.x might need to change it to **false** to continue using a sequence object or table based generator and maintain backward compatibility. The application can override this value in the **persistence.xml** file. More information on this behavior is provided below.
hibernate.transaction.jta.platform	Instance of **org.hibernate.service.jta.platform.spi.JtaPlatform** Interface	This class passes the transaction manager, user transaction, and transaction synchronization registry into Hibernate.
hibernate.ejb.resource_scanner	Instance of **org.hibernate.ejb.packaging.Scanner** Interface	This class knows how to use the JBoss EAP 6 annotation indexer to provide faster deployment.
hibernate.transaction.manager_lookup_class		This property is removed if found in the persistence.xml because it could conflict with **hibernate.transaction.jta.platform**
hibernate.session_factory_name	*QUALIFIED_PERSISTENCE_UNIT_NAME*	This is set to the application name + persistence unit name. The application can specify a different value, but it must be unique across all application deployments on the JBoss EAP 6 instance.
hibernate.session_factory_name_is_jndi	false	This is set only if the application did not specify a value for the **hibernate.session_factory_name**.
hibernate.ejb.entitymanager_factory_name	*QUALIFIED_PERSISTENCE_UNIT_NAME*	This is set to the application name + persistence unit name. The application can specify a different value but it needs to be unique across all application deployments on the JBoss EAP 6 instance.

In Hibernate 4.x, if **new_generator_mappings** is set to **true**:

- **@GeneratedValue(AUTO)** maps to **org.hibernate.id.enhanced.SequenceStyleGenerator**.
- **@GeneratedValue(TABLE)** maps to **org.hibernate.id.enhanced.TableGenerator**.

- **`@GeneratedValue(SEQUENCE)`** maps to **`org.hibernate.id.enhanced.SequenceStyleGenerator`**.

In Hibernate 4.x, if **`new_generator_mappings`** is set to **`false`**:

- **`@GeneratedValue(AUTO)`** maps to Hibernate "native".
- **`@GeneratedValue(TABLE)`** maps to **`org.hibernate.id.MultipleHiLoPerTableGenerator`**.
- **`@GeneratedValue(SEQUENCE)`** maps to Hibernate "seqhilo".

For more information about these properties, go to http://www.hibernate.org/docs and view the Hibernate 4.1 Developer Guide.

JPA persistence properties

The following JPA properties are supported in the persistence unit definition in the **`persistence.xml`** file:

Table 3.4. JPA Persistence Unit Properties

Property Name	Default Value	Purpose
`jboss.as.jpa.providerModule`	**`org.hibernate`**	The name of the persistence provider module. The value should be **`hibernate3-bundled`** if Hibernate 3 JARs are in the application archive. If a persistence provider is packaged with the application, this value should be **`application`**.
`jboss.as.jpa.adapterModule`	**`org.jboss.as.jpa.hibernate:4`**	The name of the integration classes that help JBoss EAP 6 to work with the persistence provider. Current valid values are: • **`org.jboss.as.jpa.hibernate:4`**: This is for the Hibernate 4 integration classes • **`org.jboss.as.jpa.hibernate:3`**: This is for the Hibernate 3 integration classes

Report a bug

3.2.2.4. Update Your Hibernate 3 Application to Use Hibernate 4

Summary

When you update your application to use Hibernate 4, some updates are general and apply regardless of version of Hibernate currently used by the application. For other updates, you must determine which version the application currently uses.

Procedure 3.12. Update the application to use Hibernate 4

1. The default behavior of autoincrement sequence generator has changed in JBoss EAP 6. For more information, see Section 3.2.2.5, “Preserve the Existing Behavior of the Hibernate Identity Auto Generated Value”.

2. Determine the version of Hibernate currently used by the application and choose the correct update procedure below.

    - Section 3.2.2.6, “Migrate Your Hibernate 3.3.x Application to Hibernate 4.x”
    - Section 3.2.2.7, “Migrate Your Hibernate 3.5.x Application to Hibernate 4.x”

3. See Section 3.2.2.8, “Modify Persistence Properties for Migrated Seam and Hibernate Applications that Run in a Clustered Environment” if you plan to run your application in a clustered environment.

Report a bug

3.2.2.5. Preserve the Existing Behavior of the Hibernate Identity Auto Generated Value

Hibernate 3.5 introduced a core property named **`hibernate.id.new_generator_mappings`** that directs how identity or sequence columns are generated when using **`@GeneratedValue`**. In JBoss EAP 6, the default value for this property is set as follows:

- When you deploy a native Hibernate application, the default value is **`false`**.
- When you deploy a JPA application, the default value is **`true`**.

Guidelines for New Applications

New applications that use the **`@GeneratedValue`** annotation should set the value for the **`hibernate.id.new_generator_mappings`** property to **`true`**. This is the preferred setting because it is more portable across different databases. In most cases it is more efficient and, in some cases, it addresses compatibility with the JPA 2 specification.

- For new JPA applications, JBoss EAP 6 defaults the **`hibernate.id.new_generator_mappings`** property to **`true`** and it should not be changed.
- For new native Hibernate applications, JBoss EAP 6 defaults the **`hibernate.id.new_generator_mappings`** property to **`false`**. You should set this property to **`true`**.

Guidelines for Existing JBoss EAP 5 Applications

Existing applications that use the **`@GeneratedValue`** annotation should make sure that the same generator is used to create primary key values for new entities when the application is migrated to JBoss EAP 6.

- For existing JPA applications, JBoss EAP 6 defaults the **`hibernate.id.new_generator_mappings`** property to **`true`**. You should set this property to **`false`** in the **`persistence.xml`** file.
- For existing native Hibernate applications, JBoss EAP 6 defaults the **`hibernate.id.new_generator_mappings`** to **`false`** and it should not be changed.

For more information about these property settings, see Section 3.2.2.3, “Persistence Unit Properties”.

Report a bug

3.2.2.6. Migrate Your Hibernate 3.3.x Application to Hibernate 4.x

1. **Map Hibernate `text` types to `JDBC LONGVARCHAR`**
   In versions of Hibernate prior to 3.5, **`text`** type was mapped to **`JDBC CLOB`**. A new Hibernate type, **`materialized_clob`**, was added in Hibernate 4 to map Java **`String`** properties to **`JDBC CLOB`**. If your application has properties configured as **`type="text"`** that are intended to be mapped to **`JDBC CLOB`**, you must do one of the following:

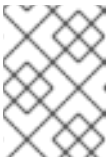
   a. If your application uses hbm mapping files, change the property to **`type="materialized_clob"`**.

   b. If your application uses annotations, you should replace **`@Type(type = "text")`** with **`@Lob`**.

2. **Review code to find changes in returned value types**
   Numeric aggregate criteria projections now return the same value type as their HQL counterparts. As a result, the return types from the following projections in **`org.hibernate.criterion`** have changed.

   a. Due to changes in **`CountProjection`**, **`Projections.rowCount()`**, **`Projections.count(propertyName)`**, and **`Projections.countDistinct(propertyName)`**, the **`count`** and **`count distinct`** projections now return a **`Long`** value.

   b. Due to changes in **`Projections.sum(propertyName)`**, the **`sum`** projections now return a value type that depends on the property type.

   NOTE

   Failure to modify your application code could result in a **`java.lang.ClassCastException`**.

      i. For properties mapped as Long, Short, Integer, or primitive integer types, a Long value is returned;

      ii. For properties mapped as Float, Double, or primitive floating point types, a Double value is returned.

Report a bug

3.2.2.7. Migrate Your Hibernate 3.5.x Application to Hibernate 4.x

1. Merge the AnnotationConfiguration into the Configuration.

   Although **`AnnotationConfiguration`** is now deprecated, it should not affect your migration.

   If you are still using an **`hbm.xml`** file, you should be aware that JBoss EAP 6 now uses the **`org.hibernate.cfg.EJB3NamingStrategy`** in **`AnnotationConfiguration`** instead of the **`org.hibernate.cfg.DefaultNamingStrategy`** that was used in previous releases. This can result in naming mismatches. If you rely on the naming strategy to default the name of an association (many-to-many and collections of elements) table, you may see this issue. To resolve it, you can tell Hibernate to use the legacy

`org.hibernate.cfg.DefaultNamingStrategy` by calling **`Configuration#setNamingStrategy`** and passing it **`org.hibernate.cfg.DefaultNamingStrategy#INSTANCE`**.

2. Modify the namespaces to conform to the new Hibernate DTD file names as noted in the following table.

Table 3.5. DTD Namespace Mapping Table

Previous DTD Namespace	New DTD Namespace
http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd	http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd
http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd	http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd

3. Modify environment variables.

   a. If you are using Oracle and using the **`materialized_clob`** or **`materialized_blob`** properties, the global environment variable **`hibernate.jdbc.use_streams_for_binary`** must be set to true.

   b. If you are using PostgreSQL and using the **`CLOB`** or **`BLOB`** properties, the global environment variable **`hibernate.jdbc.use_streams_for_binary`** must be set to false.

Report a bug

3.2.2.8. Modify Persistence Properties for Migrated Seam and Hibernate Applications that Run in a Clustered Environment

If you migrate a JPA container-managed application, the properties that influence serialization of extended persistence contexts are automatically passed to the container.

However, due to changes in Hibernate, you may run into serialization issues if you run your migrated Seam or Hibernate application in a clustered environment. You may see error log messages similar to the following:

```
javax.ejb.EJBTransactionRolledbackException: JBAS010361: Failed to
deserialize
....
Caused by: java.io.InvalidObjectException: could not resolve session
factory during session deserialization
[uuid=8aa29e74373ce3a301373ce3a44b0000, name=null]
```

To resolve these errors, you need to modify properties in the configuration file. In most cases this is the **`persistence.xml`** file. For native Hibernate API applications, this is the **`hibernate.cfg.xml`** file.

Procedure 3.13. Set persistence properties to run in a clustered environment

1. Set the **`hibernate.session_factory_name`** value to a unique name. This name must be unique across all application deployments on the JBoss EAP 6 instance. For example:

```
<property name="hibernate.session_factory_name" value="jboss-seam-
booking.ear_session_factory"/>
```

2. Set the **`hibernate.ejb.entitymanager_factory_name`** value to a unique name. This name must be unique across all application deployments on the JBoss EAP 6 instance. For example:

```
<property name="hibernate.ejb.entitymanager_factory_name"
value="seam-booking.ear_PersistenceUnitName"/>
```

For more information about the Hibernate JPA Persistence Unit Property settings, see Section 3.2.2.3, “Persistence Unit Properties”.

Report a bug

3.2.2.9. Update Your Application to Conform to the JPA 2.0 Specification

Summary

The JPA 2.0 specification requires that a persistence context cannot be propagated outside of a JTA transaction. If your application uses only transaction-scoped persistence contexts, the behavior is the same in JBoss EAP 6 as it was in previous versions of the application server and no changes are required. However, if your application uses an extended persistence context (XPC) to allow queuing or batching of data modifications, you may need to make changes to your application.

Persistence context propagation behavior

If your application has a stateful session bean, **`Bean1`**, that uses an extended persistence context, and it calls a stateless session bean, **`Bean2`**, that uses a transaction-scoped persistence context, you can expect the following behavior to occur:

- If **`Bean1`** starts a JTA transaction and makes the **`Bean2`** method invocation with the JTA transaction active, the behavior in JBoss EAP 6 is the same as previous releases and no change is necessary.
- If **`Bean1`** does not start a JTA transaction and makes the **`Bean2`** method invocation, JBoss EAP 6 does not propagate the extended persistence context into **`Bean2`**. This behavior is different than in previous releases which did propagate the extended persistence context into **`Bean2`**. If your application expects the extended persistence context to be propagated to the bean with the transactional entity manager, you need to change your application to do the invocation within an active JTA transaction.

Report a bug

3.2.2.10. Replace JPA/Hibernate Second Level Cache with Infinispan

Summary

JBoss Cache has been replaced by Infinispan for second-level cache (2LC). This requires a change to the **`persistence.xml`** file. The syntax is slightly different, depending on if you are using JPA or Hibernate second level cache. These examples assume you are using Hibernate.

This is an example of how properties for second level cache were specified in the **persistence.xml** file in JBoss EAP 5.x.

```
<property name="hibernate.cache.region.factory_class"

value="org.hibernate.cache.jbc2.JndiMultiplexedJBossCacheRegionFactory"/>
<property name="hibernate.cache.region.jbc2.cachefactory"
value="java:CacheManager"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
<property name="hibernate.cache.region.jbc2.cfg.entity" value="mvcc-
entity"/>
<property name="hibernate.cache.region_prefix" value="services"/>
```

The following steps will use this example to configure Infinispan in JBoss EAP 6.

Procedure 3.14. Modify the `persistence.xml` file to use Infinispan

1. **Configure Infinispan for a JPA application in JBoss EAP 6**
   This is how you specify properties to achieve the same configuration for a JPA application using Infinispan in JBoss EAP 6:

   ```
   <property name="hibernate.cache.use_second_level_cache"
   value="true"/>
   ```

   In addition, you need to specify a **shared-cache-mode** with a value of **ENABLE_SELECTIVE** or **ALL** as follows:

   - **ENABLE_SELECTIVE** is the default and recommended value. It means entities are not cached unless you explicitly mark them as cacheable.

     ```
     <shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
     ```

   - **ALL** means entities are always cached even if you mark them as not cacheable.

     ```
     <shared-cache-mode>ALL</shared-cache-mode>
     ```

2. **Configure Infinispan for a native Hibernate application in JBoss EAP 6**
   This is how you specify the same configuration for a native Hibernate application using Infinispan with JBoss EAP 6:

   ```
   <property name="hibernate.cache.region.factory_class"

   value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactor
   y"/>
   <property name="hibernate.cache.infinispan.cachemanager"
        value="java:jboss/infinispan/container/hibernate"/>
   <property name="hibernate.transaction.manager_lookup_class"

   value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
   <property name="hibernate.cache.use_second_level_cache"
   ```

```
value="true"/>
```

You must also add the following dependencies to the **MANIFEST.MF** file:

```
Manifest-Version: 1.0
Dependencies: org.infinispan, org.hibernate
```

For more information about Hibernate cache properties, see: Section 3.2.2.11, “Hibernate Cache Properties”.

Report a bug

3.2.2.11. Hibernate Cache Properties

Table 3.6. Properties

Property Name	Description
hibernate.cache.region.factory_class	The classname of a custom **CacheProvider**.
hibernate.cache.use_minimal_puts	Boolean. Optimizes second-level cache operation to minimize writes, at the cost of more frequent reads. This setting is most useful for clustered caches and, in Hibernate3, is enabled by default for clustered cache implementations.
hibernate.cache.use_query_cache	Boolean. Enables the query cache. Individual queries still have to be set cacheable.
hibernate.cache.use_second_level_cache	Boolean. Used to completely disable the second level cache, which is enabled by default for classes that specify a **<cache>** mapping.
hibernate.cache.query_cache_factory	The classname of a custom **QueryCache** interface. The default value is the built-in **StandardQueryCache**.
hibernate.cache.region_prefix	A prefix to use for second-level cache region names.
hibernate.cache.use_structured_entries	Boolean. Forces Hibernate to store data in the second-level cache in a more human-friendly format.
hibernate.cache.default_cache_concurrency_strategy	Setting used to give the name of the default **org.hibernate.annotations.CacheConcurrencyStrategy** to use when either **@Cacheable** or **@Cache** is used. **@Cache(strategy="..")** is used to override this default.

Report a bug

3.2.2.12. Migrate to Hibernate Validator 4

Summary

Hibernate Validator 4.x is a completely new code base that implements JSR 303 - Bean Validation. The migration process from Validator 3.x to 4.x is fairly straightforward, but there are a few changes you must make when you migrate your application.

Procedure 3.15. You may need to perform one or more of the following tasks

1. **Access the default ValidatorFactory**
   JBoss EAP 6 binds a default ValidatorFactory to the JNDI context under the name **`java:comp/ValidatorFactory`**.

2. **Understand life cycle triggered validation**
   When used in combination with Hibernate Core 4, life-cycle based validation is automatically enabled by Hibernate Core.

   a. Validation occurs on entity **`INSERT`**, **`UPDATE`**, and **`DELETE`** operations.

   b. You can configure the groups to be validated by event type using the following properties:

      - **`javax.persistence.validation.group.pre-persist`**,
      - **`javax.persistence.validation.group.pre-update`**, and
      - **`javax.persistence.validation.group.pre-remove`**.

      The values of these properties are the comma-separated, fully qualified class names of the groups to validate.

      Validation groups are a new feature of the Bean Validation specification. If you do not want to take advantage of this new feature, no changes are required when you migrate to Hibernate Validator 4.

   c. You can disable life-cycle based validation by setting the **`javax.persistence.validation.mode`** property to **none**. Other valid values for this property are **`auto`** (the default), **`callback`** and **`ddl`**.

3. **Configure your application to use manual validation**

   a. If you want to manually control validation, you can create a Validator in either of the following ways:

      - Create a **`Validator`** instance from the **`ValidatorFactory`** using the **`getValidator()`** method.
      - Inject Validator instances in your EJB, CDI bean or any other Java EE injectable resource.

   b. You can use the **`ValidatorContext`** returned by the **`ValidatorFactory.usingContext()`** to customize your Validator instance. Using this API you can configure a custom **`MessageInterpolator`**, **`TraverableResolver`** and **`ConstraintValidatorFactory`**. These interfaces are specified in the Bean Validation specification and are new to Hibernate Validator 4.

4. **Modify code to use the new Bean Validation constraints**

The new Bean level validation constraints require code changes when you migrate to Hibernate Validator 4.

a. To upgrade to Hibernate Validator 4, you must use the constraints in the following packages:

- **`javax.validation.constraints`**
- **`org.hibernate.validator.constraints`**

b. All constraints that existed in Hibernate Validator 3 are still available in Hibernate Validator 4. To use them, you need to import the specified class, and in some cases, change the name or type of the constraint parameter.

5. **Use custom constraints**
In Hibernate Validator 3, a custom constraint needed to implement the **`org.hibernate.validator.Validator`** interface. In Hibernate Validator 4, you need to implement the **`javax.validation.ConstraintValidator`** interface. This interface contains the same **`initialize()`** and **`isValid()`** methods as the previous interface, however, the method signature has changed. In addition, **`DDL`** alteration is no longer supported in Hibernate Validator 4.

Report a bug

3.2.3. JSF changes

3.2.3.1. Enable Applications To Use Older Versions of JSF

Summary

If your application uses an older version of JSF, you do not need to upgrade to JSF 2.0. Instead, you can create a **`jboss-deployment-structure.xml`** file to request that JBoss EAP 6 use JSF 1.2 rather than JSF 2.0 with your application deployment. This JBoss specific deployment descriptor is used to control class loading and is placed in the **`META-INF/`** or **`WEB-INF/`** directory of your WAR, or in the **`META-INF/`** directory of your EAR.

The following is an example of a **`jboss-deployment-structure.xml`** file that adds a dependency for the JSF 1.2 module and excludes or prevents the automatic loading of the JSF 2.0 module.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
    <deployment>
        <dependencies>
            <module name="javax.faces.api" slot="1.2" export="true"/>
            <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
        </dependencies>
    </deployment>
    <sub-deployment name="jboss-seam-booking.war">
        <exclusions>
            <module name="javax.faces.api" slot="main"/>
            <module name="com.sun.jsf-impl" slot="main"/>
        </exclusions>
        <dependencies>
            <module name="javax.faces.api" slot="1.2"/>
            <module name="com.sun.jsf-impl" slot="1.2"/>
        </dependencies>
    </sub-deployment>
</jboss-deployment-structure>
```

Report a bug

3.2.4. Web Services Changes

3.2.4.1. Web Services Changes

JBoss EAP 6 includes support for deploying JAX-WS Web Service endpoints. This support is provided by JBossWS. For more information about Web Services, refer to the chapter entitled *JAX-WS Web Services* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

JBossWS 4 includes the following changes that may impact your migration.

JBossWS API Project Changes

SPI and Common components were refactored in JBossWS 4. The following table lists API and packaging changes may affect your application migration.

Table 3.7. Size Log Handler Properties

Old JAR	Old Package	New JAR	New Package
JBossWS SPI	org.jboss.wsf.spi.annotation.*	JBossWS API	org.jboss.ws.api.annotation.*
JBossWS SPI	org.jboss.wsf.spi.binding.*	JBossWS API	org.jboss.ws.api.binding.*
JBossWS SPI	org.jboss.wsf.spi.management.recording.*	JBossWS API	org.jboss.ws.api.monitoring.*
JBossWS SPI	org.jboss.wsf.spi.tools.*	JBossWS API	org.jboss.ws.api.tools.*
JBossWS SPI	org.jboss.wsf.spi.tools.ant.*	JBossWS API	org.jboss.ws.tools.ant.*
JBossWS SPI	org.jboss.wsf.spi.tools.cmd.*	JBossWS API	org.jboss.ws.tools.cmd.*
JBossWS SPI	org.jboss.wsf.spi.util.ServiceLoader	JBossWS API	org.jboss.ws.api.util.ServiceLoader
JBossWS Common	org.jboss.wsf.common.*	JBossWS API	org.jboss.ws.common.*
JBossWS Common	org.jboss.wsf.common.handler.*	JBossWS API	org.jboss.ws.api.handler.*
JBossWS Common	org.jboss.wsf.common.addressing.*	JBossWS API	org.jboss.ws.api.addressing.*
JBossWS Common	org.jboss.wsf.common.DOMUtils	JBossWS API	org.jboss.ws.api.util.DOMUtils

Old JAR	Old Package	New JAR	New Package
JBossWS Native	org.jboss.ws.annotation.EndpointConfig	JBossWS API	org.jboss.ws.api.annotation.EndpointConfig

@WebContext Annotation

In JBossWS 3.4.x, this annotation was packaged as **`org.jboss.wsf.spi.annotation.WebContext`** in the JBossWS SPI project. In JBossWS 4.0, this annotation was moved to **`org.jboss.ws.api.annotation.WebContext`** in the JBossWS API project. If your application includes the obsolete dependency, you must replace the imports and dependencies in your application source code and compile it against the new JBossWS API JAR.

There is also a change to an attribute that is not backward compatible. The **`String[] virtualHosts`** attribute has been changed to **`String virtualHosts`**. In JBoss EAP 6, you can specify only one virtual host per deployment. If multiple webservices use the **`@WebContext`** annotation, the virtualHost value must be identical for all endpoints defined in the deployment archive.

Endpoint Configuration

JBossWS 4.0 provides integration of the JBoss Web Services stack with most of the Apache CXF project modules. The integration layer allows the use of standard webservices APIs, including JAX-WS. It also allows the use of Apache CFX advanced features on top of the JBoss EAP 6 container without requiring complex configuration or setup.

The **`webservice`** subsystem in the domain configuration of JBoss EAP 6 includes predefined endpoint configurations. You can also define your own additional endpoint configurations. The **`@org.jboss.ws.api.annotation.EndpointConfig`** annotation is used to reference a given endpoint configuration.

For more information on how to configure webservice endpoints in the JBoss server, refer to the chapter entitled *JAX-WS Web Services* in the Development Guide for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

jboss-webservices.xml Deployment Descriptor

JBossWS 4.0 introduces a new deployment descriptor to configure web services. The **`jboss-webservices.xml`** file provides additional information for the given deployment and partially replaces the obsolete **`jboss.xml`** file.

For EJB webservice deployments, the expect location of the **`jboss-webservices.xml`** descriptor file is in the **`META-INF/`** directory. For POJO and EJB webservice endpoints bundled in WAR file, the expected location of the **`jboss-webservices.xml`** file is in the **`WEB-INF/`** directory.

The following is an example of a **`jboss-webservices.xml`** descriptor file and a table describing the elements.

```
<webservices>
    <context-root>foo<context-root>
    <config-name>Standard WSSecurity Endpoint</config-name>
    <config-file>META-INF/custom.xml</config-file>
    <property>
        <name>prop.name</name>
        <value>prop.value</value>
    </property>
```

```
    <port-component>
        <ejb-name>TestService</ejb-name>
        <port-component-name>TestServicePort</port-component-name>
        <port-component-uri>/*</port-component-uri>
        <auth-method>BASIC</auth-method>
        <transport-guarantee>NONE</transport-guarantee>
        <secure-wsdl-access>true</secure-wsdl-access>
    </port-component>
    <webservice-description>
        <webservice-description-name>TestService</webservice-
description-name>
        <wsdl-publish-location>file:///bar/foo.wsdl</wsdl-publish-
location>
    </webservice-description>
</webservices>
```

Table 3.8. jboss-webservice.xml File Element Description

Element Name	Description
context-root	Used to customize the context root of the webservices deployment.
config-name config-file	Used to associate an endpoint deployment with a given endpoint configuration. Endpoint configurations are specified in the referenced configuration file or in the **`webservices`** subsystem of the domain configuration.
property	Used to set up simple property name value pairs to configure the webservice stack behavior.
port-component	Used to customize the EJB endpoint target URI or to configure security related properties.
webservice-description	Used to customize or override the webservice WSDL published location.

Report a bug

3.2.5. JAX-RS and RESTEasy Changes

3.2.5.1. Configure JAX-RS and RESTEasy Changes

JBoss EAP 6 automatically sets up RESTEasy, so you do not need to configure it yourself. Therefore, you should remove all of the existing RESTEasy configuration from your **`web.xml`** file and replace it with one of the following three options:

1. Subclass **`javax.ws.rs.core.Application`** and use the **`@ApplicationPath`** annotation.

   This is the easiest option and does not require any xml configuration. Simply subclass **`javax.ws.rs.core.Application`** in your application and annotate it with the path where you want to make your JAX-RS classes available. For example:

   -

```
@ApplicationPath("/mypath")
public class MyApplication extends Application {
}
```

In the above example, your JAX-RS resources are available in the path ***/MY_WEB_APP_CONTEXT*/mypath/**.

NOTE

Note the path should be specified as **/mypath**, not **/mypath/***. There should be no trailing forward-slash or asterisk.

2. Subclass **javax.ws.rs.core.Application** and use the **web.xml** file to set up the JAX-RS mapping.

   If you do not wish to use the **@ApplicationPath** annotation, you still need to subclass **javax.ws.rs.core.Application**. You then set up the JAX-RS mapping in the **web.xml** file. For example:

```
public class MyApplication extends Application {
}
```

```
<servlet-mapping>
   <servlet-name>com.acme.MyApplication</servlet-name>
   <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

   In the above example, your JAX-RS resources are available in the path ***/MY_WEB_APP_CONTEXT*/hello**.

NOTE

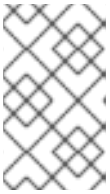
You can also use this approach to override an application path that was set using the **@ApplicationPath** annotation.

3. Modify the **web.xml** file.

   If you do not want to subclass **Application**, you can set up the JAX-RS mapping in the **web.xml** file as follows:

```
<servlet-mapping>
   <servlet-name>javax.ws.rs.core.Application</servlet-name>
   <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

   In the above example, your JAX-RS resources are available in the path ***/MY_WEB_APP_CONTEXT*/hello**.



NOTE

When you choose this option, you only need to add the mapping. You do not need to add the corresponding servlet. The server is responsible for adding the corresponding servlet automatically.

Report a bug

3.2.6. LDAP Security Realm Changes

3.2.6.1. Configure LDAP Security Realm Changes

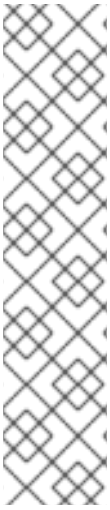
In JBoss EAP 5, the LDAP security realm was configured in an **<application-policy>** element in the **login-config.xml** file. In JBoss EAP 6, the LDAP security realm is configured in the **<security-domain>** element in the server configuration file. For a standalone server, this is the **standalone/configuration/standalone.xml** file. If you are running your server in a managed domain, this is the **domain/configuration/domain.xml** file.

The following is an example of LDAP security realm configuration in the **login-config.xml** file in JBoss EAP 5:

```
<application-policy name="mcp_ldap_domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
flag="required">
      <module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</modul
e-option>
      <module-option
name="java.naming.security.authentication">simple</module-option>
      ....
    </login-module>
  </authentication>
</application-policy>
```

This is an example of the LDAP configuration in the server configuration file in JBoss EAP 6:

```
<subsystem xmlns="urn:jboss:domain:security:1.0">
  <security-domains>
    <security-domain name="mcp_ldap_domain" cache-type="default">
      <authentication>
        <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.security.authentication"
value="simple"/>
          ...
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```



NOTE

The XML parser changed in JBoss EAP 6. In JBoss EAP 5, you specified the module options as element content like this:

```
<module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFac
tory</module-option>
```

Now, the module options must be specified as element attributes with "value=" as follows:

```
<module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
```

Report a bug

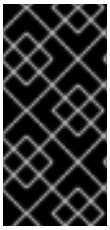
3.2.7. HornetQ Changes

3.2.7.1. About HornetQ and NFS

In most cases, NFS is not an appropriate method of storing JMS data for use with HornetQ, when using NIO as the journal type, due to the way the synchronous locking mechanism works. However, NFS can be used in certain circumstances, only on Red Hat Enterprise Linux servers. This is due to the NFS implementation used by Red Hat Enterprise Linux.

The Red Hat Enterprise Linux NFS implementation supports both direct I/O (opening files with the O_DIRECT flag set), and kernel based asynchronous I/O. With both of these features present, it is possible to use NFS as a shared storage option, under strict configuration rules:

- The Red Hat Enterprise Linux NFS client cache must be disabled.



IMPORTANT

The server log should be checked after JBoss EAP 6 is started, to ensure that the native library successfully loaded, and that the ASYNCIO journal type is being used. If the native library fails to load, HornetQ will gracefully fail to the NIO journal type, and this will be stated in the server log.

IMPORTANT

The native library that implements asynchronous I/O requires that **`libaio`** is installed on the Red Hat Enterprise Linux system where JBoss EAP 6 is running.

Report a bug

3.2.7.2. Configure a JMS Bridge to Migrate Existing JMS Messages to JBoss EAP 6

JBoss EAP 6 replaced JBoss Messaging with HornetQ as the default JMS implementation. The easiest way to migrate JMS messages from one environment to the other is to use a JMS bridge. The function of a JMS bridge is to consume messages from a source JMS destination, and send them to a target JMS destination. You can configure and deploy a JMS bridge to a JBoss EAP 5.x server or to JBoss EAP 6.1 or later server. The following procedures describe how to do this.

Procedure 3.16. Configure the JMS Bridge deployed to a JBoss EAP 5.x Server

To avoid conflicts in classes between releases, you must use the following procedure to configure the JMS bridge on JBoss EAP 5.x. The names of the SAR directory and bridge are arbitrary and can be changed if you prefer.

1. Create a subdirectory in the JBoss EAP 5 deploy directory to contain the SAR, for example: ***EAP5_HOME*/server/*PROFILE_NAME*/deploy/myBridge.sar**.

2. Create a subdirectory named **META-INF** in ***EAP5_HOME*/server/*PROFILE_NAME*/deploy/myBridge.sar/**.

3. Create a **jboss-service.xml** file that contains information similar to the following in the ***EAP5_HOME*/server/*PROFILE_NAME*/deploy/myBridge.sar/META-INF/** directory.

```
<server>
   <loader-repository>
      com.example:archive=unique-archive-name
      <loader-repository-config>java2ParentDelegation=false</loader-
repository-config>
   </loader-repository>

   <!-- JBoss EAP 6 JMS Provider -->
   <mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=EnterpriseAppli
cationPlatform6JMSProvider">
      <attribute
name="ProviderName">EnterpriseApplicationPlatform6JMSProvider</attri
bute>
      <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapter</
attribute>
      <attribute
name="FactoryRef">jms/RemoteConnectionFactory</attribute>
      <attribute
name="QueueFactoryRef">jms/RemoteConnectionFactory</attribute>
      <attribute
name="TopicFactoryRef">jms/RemoteConnectionFactory</attribute>
      <attribute name="Properties">

java.naming.factory.initial=org.jboss.naming.remote.client.InitialCo
ntextFactory

java.naming.provider.url=remote://EnterpriseApplicationPlatform6host
:4447
         java.naming.security.principal=jbossuser
         java.naming.security.credentials=jbosspass
      </attribute>
   </mbean>

   <mbean code="org.jboss.jms.server.bridge.BridgeService"
name="jboss.jms:service=Bridge,name=MyBridgeName" xmbean-
dd="xmdesc/Bridge-xmbean.xml">
      <depends optional-attribute-
name="SourceProviderLoader">jboss.messaging:service=JMSProviderLoade
r,name=JMSProvider</depends>
```

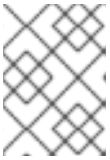
```
      <depends optional-attribute-
name="TargetProviderLoader">jboss.messaging:service=JMSProviderLoade
r,name=EnterpriseApplicationPlatform6JMSProvider</depends>
      <attribute name="SourceDestinationLookup">/queue/A</attribute>
      <attribute
name="TargetDestinationLookup">jms/queue/test</attribute>
      <attribute name="QualityOfServiceMode">1</attribute>
      <attribute name="MaxBatchSize">1</attribute>
      <attribute name="MaxBatchTime">-1</attribute>
      <attribute name="FailureRetryInterval">60000</attribute>
      <attribute name="MaxRetries">-1</attribute>
      <attribute name="AddMessageIDInHeader">false</attribute>
      <attribute name="TargetUsername">jbossuser</attribute>
      <attribute name="TargetPassword">jbosspass</attribute>
   </mbean>
</server>
```

NOTE

The **<load-repository>** is present to ensure the SAR has an isolated classloader. Also note both the JNDI look-up and the bridge "target" include security credentials for user "jbossuser" with password "jbosspass". This is because JBoss EAP 6 is secured by default. The user named "jbossuser" with password "jbosspass" was created in the **ApplicationRealm** with the **guest** role using the ***EAP_HOME*/bin/add_user.sh** script.

4. Copy the following JARs from the ***EAP_HOME*/modules/system/layers/base/** directory into the ***EAP5_HOME*/server/*PROFILE_NAME*/deploy/myBridge.sar/** directory. Replace each *VERSION_NUMBER* with the actual version number in your JBoss EAP 6 distribution.
   - **org/hornetq/main/hornetq-core-*VERSION_NUMBER*.jar**
   - **org/hornetq/main/hornetq-jms-*VERSION_NUMBER*.jar**
   - **org/jboss/ejb-client/main/jboss-ejb-client-*VERSION_NUMBER*.jar**
   - **org/jboss/logging/main/jboss-logging-*VERSION_NUMBER*.jar**
   - **org/jboss/logmanager/main/jboss-logmanager-*VERSION_NUMBER*.jar**
   - **org/jboss/marshalling/main/jboss-marshalling-*VERSION_NUMBER*.jar**
   - **org/jboss/marshalling/river/main/jboss-marshalling-river-*VERSION_NUMBER*.jar**
   - **org/jboss/remote-naming/main/jboss-remote-naming-*VERSION_NUMBER*.jar**
   - **org/jboss/remoting3/main/jboss-remoting-*VERSION_NUMBER*.jar**
   - **org/jboss/sasl/main/jboss-sasl-*VERSION_NUMBER*.jar**
   - **org/jboss/netty/main/netty-*VERSION_NUMBER*.jar**

- **org/jboss/remoting3/remote-jmx/main/remoting-jmx-*VERSION_NUMBER*.jar**
- **org/jboss/xnio/main/xnio-api-*VERSION_NUMBER*.jar**
- **org/jboss/xnio/nio/main.xnio-nio-*VERSION_NUMBER*.jar**



NOTE

Do not simply copy the ***EAP_HOME*/bin/client/jboss-client.jar** because the javax API classes will conflict with those in JBoss EAP 5.x.

Procedure 3.17. Configure the JMS Bridge deployed to a JBoss EAP 6.x Server

In JBoss EAP 6.1 and later, the JMS bridge can be used to bridge messages from any JMS 1.1 compliant server. Because the source and target JMS resources are looked up using JNDI, the JNDI lookup classes of the source messaging provider, or message broker, must be bundled in a JBoss Module. The following procedure uses the fictitious 'MyCustomMQ' message broker as an example.

1. Create the JBoss module for the messaging provider.

   a. Create a directory structure under ***EAP_HOME*/modules/system/layers/base/** for the new module. The **main/** subdirectory will contain the client JARs and **module.xml** file. The following is an example of the directory structure created for the MyCustomMQ messaging provider: ***EAP_HOME*/modules/system/layers/base/org/mycustommq/main/**

   b. In the **main/** subdirectory, create a **module.xml** file containing the module definition for the messaging provider. The following is an example of the **module.xml** created for the MyCustomMQ messaging provider.

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.mycustommq">
    <properties>
        <property name="jboss.api" value="private"/>
    </properties>

    <resources>
        <!-- Insert resources required to connect to the source
or target   -->
        <resource-root path="mycustommq-1.2.3.jar" />
        <resource-root path="mylogapi-0.0.1.jar" />
    </resources>

    <dependencies>
       <!-- Add the dependencies required by JMS Bridge code
-->
       <module name="javax.api" />
       <module name="javax.jms.api" />
       <module name="javax.transaction.api"/>
       <!-- Add a dependency on the org.hornetq module since we
send         -->
       <!-- messages tothe HornetQ server embedded in the local
EAP instance -->
       <module name="org.hornetq" />
```

```
    </dependencies>
</module>
```

c. Copy the messaging provider JARs required for the JNDI lookup of the source resources to the module's **main/** subdirectory. The directory structure for the MyCustomMQ module should now look like the following.

```
modules/
`-- system
    `-- layers
        `-- base
            `-- org
                `-- mycustommq
                    `-- main
                        |-- mycustommq-1.2.3.jar
                        |-- mylogapi-0.0.1.jar
                        |-- module.xml
```

2. Configure the JMS bridge in the **messaging** subsystem of the JBoss EAP 6 server.

a. Before you begin, stop the server and back up the current server configuration files. If you are running a standalone server, this is the ***EAP_HOME*/standalone/configuration/standalone-full-ha.xml** file. If you are running a managed domain, back up both the ***EAP_HOME*/domain/configuration/domain.xml** and the ***EAP_HOME*/domain/configuration/host.xml** files.

b. Add the **<jms-bridge>** element to the **messaging** subsystem in the server configuration file. The **<source>** and **<target>** elements provide the names of the JMS resources used for JNDI lookups. If **<user>** and **<password>** credentials are specified, they are passed as arguments when JMS connection is created.

The following is an example of the **<jms-bridge>** element configured for the MyCustomMQ messaging provider:

```
<subsystem xmlns="urn:jboss:domain:messaging:1.3">
   ...
   <jms-bridge name="myBridge" module="org.mycustommq">
      <source>
         <connection-factory name="ConnectionFactory"/>
         <destination name="sourceQ"/>
         <user>user1</user>
         <password>pwd1</password>
         <context>
            <property key="java.naming.factory.initial"
value="org.mycustommq.jndi.MyCustomMQInitialContextFactory"/>
            <property key="java.naming.provider.url"
value="tcp://127.0.0.1:9292"/>
         </context>
      </source>
      <target>
         <connection-factory name="java:/ConnectionFactory"/>
         <destination name="/jms/targetQ"/>
      </target>
```

```
        <quality-of-service>DUPLICATES_OK</quality-of-service>
        <failure-retry-interval>500</failure-retry-interval>
        <max-retries>1</max-retries>
        <max-batch-size>500</max-batch-size>
        <max-batch-time>500</max-batch-time>
        <add-messageID-in-header>true</add-messageID-in-header>
    </jms-bridge>
</subsystem>
```

In the above example, the JNDI properties are defined in the **<context>** element for the **<source>**. If the **<context>** element is omitted, as in the **<target>** example above, the JMS resources are looked up in the local instance.

Report a bug

3.2.7.3. Migrate Your Application to Use HornetQ as the JMS Provider

JBoss Messaging is no longer included in JBoss EAP 6. If your application uses JBoss Messaging as the messaging provider, you need to replace the JBoss Messaging code with HornetQ.

Procedure 3.18. Before you start

1. Shut down the client and server.
2. Make a backup copy of any JBoss Messaging data. The message data is stored in a database in tables prefixed with **JBM**_.

Procedure 3.19. Change your provider to HornetQ

1. **Transfer configurations**
   Transfer the existing JBoss Messaging configurations to the JBoss EAP 6 configuration. The following configurations can be found in deployment descriptors located on the JBoss Messaging server:
   - Connection Factories Service Configuration

     This configuration describes the JMS connection factories deployed with the JBoss Messaging server. JBoss Messaging configures connection factories in a file named **connection-factories-service.xml** which is located in the deployment directory of the application server.
   - Destination Configuration

     This configuration describes JMS queues and topics deployed with JBoss Messaging server. By default, JBoss Messaging configures destinations in a file named **destinations-service.xml** which is located in the deployment directory of the application server.
   - Message Bridge Service Configuration

     This configuration includes bridge services deployed with JBoss Messaging server. No bridges are deployed by default so the name of the deployment file varies depending on your JBoss Messaging installation.
2. **Modify your application code**

If the application code uses standard JMS, no code changes are required. However, if the application will be connecting to a cluster, you must carefully review the HornetQ documentation on clustering semantics. Clustering is outside the scope of the JMS specification and HornetQ and JBoss Messaging have taken substantially different approaches in their respective implementations of clustering functionality.

If the application uses features specific to JBoss Messaging, you must modify the code to use the equivalent features available in HornetQ.

For more information on how to configure messaging with HornetQ, see: Section 3.2.7.4, “Configure Messaging with HornetQ”

3. **Migrate existing messages**
   Move any messages in the JBoss Messaging database to the HornetQ journal using a JMS bridge. Instructions for configuring the JMS bridge can be found here: Section 3.2.7.2, “Configure a JMS Bridge to Migrate Existing JMS Messages to JBoss EAP 6”.

Report a bug

3.2.7.4. Configure Messaging with HornetQ

The recommended method of configuring messaging in JBoss EAP 6 is in either the Management Console or Management CLI. You can make persistent changes with either of these management tools without needing to manually edit the **`standalone.xml`** or **`domain.xml`** configuration files. It is useful however to familiarize yourself with the messaging components of the default configuration files, where documentation examples using management tools give configuration file snippets for reference.

Report a bug

3.2.8. Clustering Changes

3.2.8.1. Make Changes to Your Application for Clustering

1. **Start JBoss EAP 6 with clustering enabled**
   To enable clustering in JBoss EAP 5.x, you needed to start your server instances using the **`all`** profile or some derivation of it, like this:

   ```
   $ EAP5_HOME/bin/run.sh -c all
   ```

   In JBoss EAP 6, the method for enabling clustering depends on whether the servers are standalone or running in a managed domain.

   a. **Enable clustering for servers running in a managed domain**
      To enable clustering for servers started using the domain controller, update your **`domain.xml`** and designate a server group to use the **`ha`** profile and **`ha-sockets`** socket binding group. For example:

      ```
      <server-groups>
        <server-group name="main-server-group" profile="ha">
          <jvm name="default">
            <heap size="64m" max-size="512m"/>
          </jvm>
          <socket-binding-group ref="ha-sockets"/>
        </server-group>
      </server-group>
      ```

b. **Enable clustering for standalone servers**
To enable clustering for standalone servers, start the server using the appropriate configuration file as follows: **`$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -Djboss.node.name=UNIQUE_NODE_NAME`**

2. **Specify the bind address**
In JBoss EAP 5.x, you would typically indicate the bind address used for clustering using the **`-b`** command line argument like this: **`$ EAP_HOME/bin/run.sh -c all -b 192.168.0.2`**

In JBoss EAP 6, bind addresses are explicitly defined by the relevant socket bindings within the JBoss EAP 6 configuration files. For servers started using the domain controller, bind addresses are specified within the **`domain/configuration/host.xml`** file. For standalone servers, bind addresses are specified within the **`standalone-ha.xml`** file:

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.0.2"/>
      </interface>
      <interface name="public">
    <inet-address value="192.168.0.2"/>
  </interface>
</interfaces>
```

```
<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
interface="public">
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```

In the example above, the **`public`** interface is specified as the default interface for all sockets within the **`ha-sockets`** socket binding group.

3. **Configure `jvmRoute` to support mod_jk and mod_proxy**
In JBoss EAP 5, the web server **`jvmRoute`** was configured using a property in the **`server.xml`** file. In JBoss EAP 6, the **`jvmRoute`** attribute is configured in the web subsystem of the server configuration file using the **`instance-id`** attribute as follows:

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
server="default-host" native="false" instance-id="
{JVM_ROUTE_SERVER}">
```

The *{JVM_ROUTE_SERVER}* above should be replaced by the jvmRoute server ID.

The **`instance-id`** can also be set using the Management Console.

4. **Specify the multicast address and port**
In JBoss EAP 5.x, you could specify the multicast address and port used for intra-cluster communication using the command line arguments **`-u`** and **`-m`**, respectively, like this: **`$ EAP_HOME/bin/run.sh -c all -u 228.11.11.11 -m 45688`**

In JBoss EAP 6, the multicast address and port used for intra-cluster communication are defined by the socket-binding referenced by the relevant JGroups protocol stack as follows:

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
    <stack name="udp">
        <transport type="UDP" socket-binding="jgroups-udp"/>
        <!-- ... -->
    </stack>
</subsystem>
```

```
<socket-binding-groups>
    <socket-binding-group name="ha-sockets" default-
interface="public">
        <!-- ... -->
        <socket-binding name="jgroups-udp" port="55200" multicast-
address="228.11.11.11" multicast-port="45688"/>
        <!-- ... -->
    </socket-binding-group>
</socket-binding-groups>
```

If you prefer to specify the multicast address and port in the command line, you can define the multicast address and ports as system properties and then use those properties on the command line when you start the server. In the following example, **`jboss.mcast.addr`** is the variable name for the multicast address and **`jboss.mcast.port`** is the variable name for the port.

```
<socket-binding name="jgroups-udp" port="55200"
 multicast-address="${jboss.mcast.addr:230.0.0.4}" multicast-
port="${jboss.mcast.port:45688}"/>
```

You can then start your server using the following command line arguments: **`$ EAP_HOME/bin/domain.sh -Djboss.mcast.addr=228.11.11.11 -Djboss.mcast.port=45688`**

5. **Use an alternate protocol stack**
   In JBoss EAP 5.x, you could manipulate the default protocol stack used for all clustering services using the **`jboss.default.jgroups.stack`** system property. **`$ EAP_HOME/bin/run.sh -c all -Djboss.default.jgroups.stack=tcp`**

   In JBoss EAP 6, the default protocol stack is defined by the JGroups subsystem within **`domain.xml`** or **`standalone-ha.xml`**:

   ```
   <subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
       <stack name="udp">
           <!-- ... -->
       </stack>
   </subsystem>
   ```

6. **Replace Buddy Replication**
   JBoss EAP 5.x used JBoss Cache Buddy Replication to suppress replication of data to all instances in a cluster. It required that you pass the argument **`-Djboss.cluster.buddyRepl`** on the command line when you started the JBoss server.

   In JBoss EAP 6, Buddy Replication has been replaced by Infinispan's far superior distributed cache or DIST mode. Distribution is a powerful clustering mode which allows Infinispan to scale

linearly as more servers are added to the cluster. The following is an example of how to configure the server to use the DIST caching mode.

a. Open a command line and start the server with either the HA or Full Profile, for example:

```
EAP_HOME/bin/standalone.sh -c standalone-ha.xml
```

b. Open another command line and connect to the Management CLI.

- For Linux, enter the following at the command line:

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

- For Windows, enter the following at a command line:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

You should see the following response:

```
Connected to standalone controller at localhost:9999
```

c. Issue the following commands:

```
/subsystem=infinispan/cache-container=web/:write-
attribute(name=default-cache,value=dist)
/subsystem=infinispan/cache-container=web/distributed-
cache=dist/:write-attribute(name=owners,value=3)
:reload
```

You should see the following response after each command:

```
"outcome" => "success"
```

These commands create the following configuration in the **`infinispan`** subsystem of the **`standalone-ha.xml`** file:

```
<cache-container name="web" aliases="standard-session-cache"
default-cache="dist"
module="org.jboss.as.clustering.web.infinispan">
    <transport lock-timeout="60000"/>
    <replicated-cache name="repl" mode="ASYNC" batching="true">
        <file-store/>
    </replicated-cache>
    <replicated-cache name="sso" mode="SYNC" batching="true"/>
    <distributed-cache name="dist" owners="3" l1-lifespan="0"
mode="ASYNC" batching="true">
        <file-store/>
    </distributed-cache>
</cache-container>
```

For more information, refer to the chapter entitled *Clustering in Web Applications* in the *Development Guide* for JBoss EAP 6 located on the Customer Portal at https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Report a bug

3.2.8.2. Implement an HA Singleton

Summary

In JBoss EAP 5, HA singleton archives were deployed in the **`deploy-hasingleton/`** directory separate from other deployments. This was done to prevent automatic deployment and to ensure the HASingletonDeployer service controlled the deployment and deployed the archive only on the master node in the cluster. There was no hot deployment feature, so redeployment required a server restart. Also, if the master node failed requiring another node to take over as master, the singleton service had to go through the entire deployment process in order to provide the service.

In JBoss EAP 6 this has changed. Using a SingletonService, the target service is installed on every node in the cluster but is only started on one node at any given time. This approach simplifies the deployment requirements and minimizes the time required to relocate the singleton master service between nodes.

Procedure 3.20. Implement an HA Singleton Service

1. **Write the HA singleton service application.**
   The following is a simple example of a Service that is wrapped with the SingletonService decorator to be deployed as a singleton service.

   a. **Create a singleton service.**
      The following listing is an example of a singleton service:

```
package com.mycompany.hasingleton.service.ejb;

import java.util.concurrent.atomic.AtomicBoolean;
import java.util.logging.Logger;

import org.jboss.as.server.ServerEnvironment;
import org.jboss.msc.inject.Injector;
import org.jboss.msc.service.Service;
import org.jboss.msc.service.ServiceName;
import org.jboss.msc.service.StartContext;
import org.jboss.msc.service.StartException;
import org.jboss.msc.service.StopContext;
import org.jboss.msc.value.InjectedValue;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
public class EnvironmentService implements Service<String> {
    private static final Logger LOGGER =
Logger.getLogger(EnvironmentService.class.getCanonicalName());
    public static final ServiceName SINGLETON_SERVICE_NAME =
ServiceName.JBOSS.append("quickstart", "ha", "singleton");
    /**
     * A flag whether the service is started.
     */
    private final AtomicBoolean started = new
```

```
AtomicBoolean(false);

    private String nodeName;

    private final InjectedValue<ServerEnvironment> env = new
InjectedValue<ServerEnvironment>();

    public Injector<ServerEnvironment> getEnvInjector() {
        return this.env;
    }

    /**
     * @return the name of the server node
     */
    public String getValue() throws IllegalStateException,
IllegalArgumentException {
        if (!started.get()) {
            throw new IllegalStateException("The service '" +
this.getClass().getName() + "' is not ready!");
        }
        return this.nodeName;
    }

    public void start(StartContext arg0) throws StartException {
        if (!started.compareAndSet(false, true)) {
            throw new StartException("The service is still
started!");
        }
        LOGGER.info("Start service '" + this.getClass().getName()
+ "'");
        this.nodeName = this.env.getValue().getNodeName();
    }

    public void stop(StopContext arg0) {
        if (!started.compareAndSet(true, false)) {
            LOGGER.warning("The service '" +
this.getClass().getName() + "' is not active!");
        } else {
            LOGGER.info("Stop service '" +
this.getClass().getName() + "'");
        }
    }
}
```

b. **Create a singleton EJB to start the service as a SingletonService at server start.** The following listing is an example of a singleton EJB that starts a SingletonService on server start:

```
package com.mycompany.hasingleton.service.ejb;

import java.util.Collection;
import java.util.EnumSet;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
```

```
import javax.ejb.Singleton;
import javax.ejb.Startup;

import org.jboss.as.clustering.singleton.SingletonService;
import org.jboss.as.server.CurrentServiceContainer;
import org.jboss.as.server.ServerEnvironment;
import org.jboss.as.server.ServerEnvironmentService;
import org.jboss.msc.service.AbstractServiceListener;
import org.jboss.msc.service.ServiceController;
import org.jboss.msc.service.ServiceController.Transition;
import org.jboss.msc.service.ServiceListener;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


/**
 * A Singleton EJB to create the SingletonService during startup.
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
@Singleton
@Startup
public class StartupSingleton {
  private static final Logger LOGGER =
LoggerFactory.getLogger(StartupSingleton.class);

  /**
   * Create the Service and wait until it is started.<br/>
   * Will log a message if the service will not start in 10sec.
   */
  @PostConstruct
  protected void startup() {
    LOGGER.info("StartupSingleton will be initialized!");

    EnvironmentService service = new EnvironmentService();
    SingletonService<String> singleton = new
SingletonService<String>(service,
EnvironmentService.SINGLETON_SERVICE_NAME);
    // if there is a node where the Singleton should deployed the
election policy might set,
    // otherwise the JGroups coordinator will start it
    //singleton.setElectionPolicy(new
PreferredSingletonElectionPolicy(new
NamePreference("node2/cluster"), new
SimpleSingletonElectionPolicy()));
    ServiceController<String> controller =
singleton.build(CurrentServiceContainer.getServiceContainer())
        .addDependency(ServerEnvironmentService.SERVICE_NAME,
ServerEnvironment.class, service.getEnvInjector())
        .install();

    controller.setMode(ServiceController.Mode.ACTIVE);
    try {
      wait(controller, EnumSet.of(ServiceController.State.DOWN,
ServiceController.State.STARTING), ServiceController.State.UP);
      LOGGER.info("StartupSingleton has started the Service");
```

```
    } catch (IllegalStateException e) {
      LOGGER.warn("Singleton Service {} not started, are you sure
to start in a cluster (HA)
environment?",EnvironmentService.SINGLETON_SERVICE_NAME);
    }
  }

  /**
   * Remove the service during undeploy or shutdown
   */
  @PreDestroy
  protected void destroy() {
    LOGGER.info("StartupSingleton will be removed!");
    ServiceController<?> controller =
CurrentServiceContainer.getServiceContainer().getRequiredService(
EnvironmentService.SINGLETON_SERVICE_NAME);
    controller.setMode(ServiceController.Mode.REMOVE);
    try {
      wait(controller, EnumSet.of(ServiceController.State.UP,
ServiceController.State.STOPPING, ServiceController.State.DOWN),
ServiceController.State.REMOVED);
    } catch (IllegalStateException e) {
      LOGGER.warn("Singleton Service {} has not be stopped
correctly!",EnvironmentService.SINGLETON_SERVICE_NAME);
    }
  }

  private static <T> void wait(ServiceController<T> controller,
Collection<ServiceController.State> expectedStates,
ServiceController.State targetState) {
    if (controller.getState() != targetState) {
      ServiceListener<T> listener = new
NotifyingServiceListener<T>();
      controller.addListener(listener);
      try {
        synchronized (controller) {
          int maxRetry = 2;
          while (expectedStates.contains(controller.getState())
&& maxRetry > 0) {
            LOGGER.info("Service controller state is {}, waiting
for transition to {}", new Object[] {controller.getState(),
targetState});
            controller.wait(5000);
            maxRetry--;
          }
        }
      } catch (InterruptedException e) {
        LOGGER.warn("Wait on startup is interrupted!");
        Thread.currentThread().interrupt();
      }
      controller.removeListener(listener);
      ServiceController.State state = controller.getState();
      LOGGER.info("Service controller state is now {}",state);
      if (state != targetState) {
        throw new IllegalStateException(String.format("Failed to
wait for state to transition to %s.  Current state is %s",
```

```
targetState, state), controller.getStartException());
      }
    }
  }

  private static class NotifyingServiceListener<T> extends
AbstractServiceListener<T> {
    @Override
    public void transition(ServiceController<? extends T>
controller, Transition transition) {
      synchronized (controller) {
        controller.notify();
      }
    }
  }
}
```

c. **Create a Stateless Session Bean to access the service from a client.**
The following is an example of a stateless session bean that accesses the service from a client:

```
package com.mycompany.hasingleton.service.ejb;

import javax.ejb.Stateless;

import org.jboss.as.server.CurrentServiceContainer;
import org.jboss.msc.service.ServiceController;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * A simple SLSB to access the internal SingletonService.
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
@Stateless
public class ServiceAccessBean implements ServiceAccess {
    private static final Logger LOGGER =
LoggerFactory.getLogger(ServiceAccessBean.class);

    public String getNodeNameOfService() {
        LOGGER.info("getNodeNameOfService() is called()");
        ServiceController<?> service =
CurrentServiceContainer.getServiceContainer().getService(
                EnvironmentService.SINGLETON_SERVICE_NAME);
        LOGGER.debug("SERVICE {}", service);
        if (service != null) {
            return (String) service.getValue();
        } else {
            throw new IllegalStateException("Service '" +
EnvironmentService.SINGLETON_SERVICE_NAME + "' not found!");
        }
    }
}
```

d. **Create the business logic interface for the SingletonService.**
The following is an example of a business logic interface for the SingletonService:

```
package com.mycompany.hasingleton.service.ejb;

import javax.ejb.Remote;

/**
 * Business interface to access the SingletonService via this EJB
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
@Remote
public interface ServiceAccess {
    public abstract String getNodeNameOfService();
}
```

2. **Start each JBoss EAP 6 instance with clustering enabled.**

The method for enabling clustering depends on whether the servers are standalone or running in a managed domain.

a. **Enable clustering for servers running in a managed domain.**
You can enable clustering using the Management CLI or you can manually edit the configuration file.

- **Enable clustering using the Management CLI.**

i. **Start your domain controller.**

ii. **Open a command prompt for your operating system.**

iii. **Connect to the Management CLI passing the domain controller IP address or DNS name.**
In this example, assume the IP address of the domain controller is **`192.168.0.14`**.

- For Linux, enter the following at the command line:

```
$ EAP_HOME/bin/jboss-cli.sh --connect --
controller=192.168.0.14
```

- For Windows, enter the following at a command line:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect --
controller=192.168.0.14
```

You should see the following response:

```
Connected to domain controller at 192.168.0.14
```

iv. **Add the `main-server` server group.**

```
[domain@192.168.0.14:9999 /] /server-group=main-server-
group:add(profile="ha",socket-binding-group="ha-sockets")
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

v. **Create a server named `server-one` and add it to the `main-server` server group.**

```
[domain@192.168.0.14:9999 /]  /host=station14Host2/server-
config=server-one:add(group=main-server-group,auto-
start=false)
{
    "outcome" => "success",
    "result" => undefined
}
```

vi. **Configure the JVM for the `main-server` server group.**

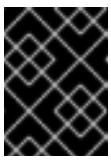
```
[domain@192.168.0.14:9999 /] /server-group=main-server-
group/jvm=default:add(heap-size=64m,max-heap-size=512m)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

vii. **Create a server named `server-two`, put it in a separate server group, and set its port offset to 100.**

```
[domain@192.168.0.14:9999 /]  /host=station14Host2/server-
config=server-two:add(group=distinct2,socket-binding-port-
offset=100)
{
    "outcome" => "success",
    "result" => undefined
}
```

- **Enable clustering by manually editing the server configuration files.**

  i. **Stop the JBoss EAP 6 server.**

  IMPORTANT

  You must stop the server before editing the server configuration file for your change to be persisted on server restart.

  ii. **Open the `domain.xml` configuration file for editing**
  Designate a server group to use the **ha** profile and **`ha-sockets`** socket binding group as follows:

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-groups>
```

iii. **Open the `host.xml` configuration file for editing**
Modify the file as follows:

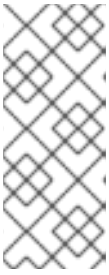
```
<servers>
  <server name="server-one" group="main-server-group" auto-
start="false"/>
  <server name="server-two" group="distinct2">
    <socket-bindings port-offset="100"/>
  </server>
<servers>
```

iv. **Start the server.**

- For Linux, type: ***EAP_HOME*`/bin/domain.sh`**
- For Microsoft Windows, type: ***EAP_HOME*`\bin\domain.bat`**

b. **Enable clustering for standalone servers**
To enable clustering for standalone servers, start the server using the node name and the **`standalone-ha.xml`** configuration file as follows:

- For Linux, type: ***EAP_HOME*`/bin/standalone.sh --server-config=standalone-ha.xml -Djboss.node.name=`*UNIQUE_NODE_NAME***
- For Microsoft Windows, type: ***EAP_HOME*`\bin\standalone.bat --server-config=standalone-ha.xml -Djboss.node.name=`*UNIQUE_NODE_NAME***



NOTE

To avoid port conflicts when running multiple servers on one machine, configure the **`standalone-ha.xml`** file for each server instance to bind on a separate interface. Alternatively, you can start subsequent server instances with a port offset using an argument like the following on the command line: **`-Djboss.socket.binding.port-offset=100`**.

3. **Deploy the application to the servers**
If you use Maven to deploy your application, use the following Maven command to deploy to the server running on the default ports:

```
mvn clean install jboss-as:deploy
```

To deploy to additional servers, pass the server name and port number on the command line:

```
mvn clean package jboss-as:deploy -Ddeploy.hostname=localhost -Djboss-
as.port=10099
```

Report a bug

3.2.9. Service-style Deployment Changes

3.2.9.1. Update Applications That Use Service-style Deployments

Summary

Although JBoss EAP 6 no longer uses service-style descriptors, the container supports these service-style deployments without change where possible. This means that if you used **`jboss-service.xml`** or **`jboss-beans.xml`** deployment descriptors in your JBoss EAP 5.x application, they should run with little or no modification in JBoss EAP 6. You can continue to package the files in the EAR or SAR, or you can place the files directly in the deployments directory. If you are running a standalone server, the deployments directory is located here: ***`EAP_HOME`*`/standalone/deployments/`**. If you are running a managed domain, you must use the console or CLI to deploy the application.

Report a bug

3.2.10. Remote Invocation Changes

3.2.10.1. Migrate JBoss EAP 5 Deployed Applications That Make Remote Invocations to JBoss EAP 6

Summary

In JBoss EAP 6, there are two ways to make remote invocations to the server:

- You can use the new JBoss specific EJB client API to do the invocation.
- You can use JNDI to lookup a proxy for your bean and invoke on that returned proxy.

This section covers option 2: coding changes required for clients that use JNDI.

In JBoss EAP 5, the EJB remote interface was bound in JNDI, by default, under the name "ejbName/local" for local interfaces, and "ejbName/remote" for the remote interfaces. The client application then looked up the bean using "ejbName/remote".

In JBoss EAP 6, you use the **`ejb:`***NAMESPACE_NAME* for remote access to EJBs with the following syntax: For stateless beans:

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>
```

For stateful beans:

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>?stateful
```

The values to be substituted in the above syntax are:

- **<app-name>** - the application name of the deployed EJBs. This is typically the ear name without the .ear suffix, however, the name can be overridden in the application.xml file. If the application is not deployed as a .ear, this value is an empty string. Assume this example was not deployed as an EAR.
- **<module-name>** - the module name of the deployed EJBs on the server. This is typically the jar name of the EJB deployment, without the .jar suffix, but can be overridden using the ejb-jar.xml. In this example, assume the EJBs were deployed in a jboss-as-ejb-remote-app.jar, so the module name is jboss-as-ejb-remote-app.
- **<distinct-name>** - an optional distinct name for the EJB. This example does not use a distinct name, so it uses an empty string.
- **<bean-name>** - by default, is the simple class name of the bean implementation class.
- **<fully-qualified-classname-of-the-remote-interface>** - the remote view fully qualified class name.

Update the client code

Assume you have deployed the following stateless EJB to a JBoss EAP 6 server. Note that it exposes a remote view for the bean.

```
@Stateless
@Remote(RemoteCalculator.class)
public class CalculatorBean implements RemoteCalculator {

    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

In JBoss EAP 5, the client EJB lookup and invocation was coded something like this:

```
InitialContext ctx = new InitialContext();
RemoteCalculator calculator = (RemoteCalculator)
ctx.lookup("CalculatorBean/remote");
int a = 204;
int b = 340;
int sum = calculator.add(a, b);
```

In JBoss EAP 6, using the information described above, the client lookup and invocation is coded like this:

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
final String appName = "";
final String moduleName = "jboss-as-ejb-remote-app";
```

```
final String distinctName = "";
final String beanName = CalculatorBean.class.getSimpleName();
final String viewClassName = RemoteCalculator.class.getName();
final RemoteCalculator statelessRemoteCalculator =  (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName);

int a = 204;
int b = 340;
int sum = statelessRemoteCalculator.add(a, b);
```

If your client is accessing a stateful EJB, you must append "?stateful" to the end of the context lookup like this:

```
final RemoteCalculator statefulRemoteCalculator =  (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName + "?stateful")
```

A complete working example, including both server and client code, can be found in the Quickstarts. For more information, refer to *Review the Quickstart Tutorials* in the chapter entitled *Get Started Developing Applications* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

For more information on remote invocations using JNDI, refer to Section 3.2.10.2, “Invoke a Session Bean Remotely using JNDI” .

Report a bug

3.2.10.2. Invoke a Session Bean Remotely using JNDI

This task describes how to add support to a remote client for the invocation of session beans using JNDI. The task assumes that the project is being built using Maven.

The **`ejb-remote`** quickstart contains working Maven projects that demonstrate this functionality. The quickstart contains projects for both the session beans to deploy and the remote client. The code samples below are taken from the remote client project.

This task assumes that the session beans do not require authentication.

Prerequisites

The following prerequisites must be satisfied before beginning:

- You must already have a Maven project created ready to use.
- Configuration for the JBoss EAP 6 Maven repository has already been added.
- The session beans that you want to invoke are already deployed.
- The deployed session beans implement remote business interfaces.
- The remote business interfaces of the session beans are available as a Maven dependency. If the remote business interfaces are only available as a JAR file then it is recommended to add the JAR to your Maven repository as an artifact. Refer to the Maven documentation for the **`install:install-file`** goal for directions, http://maven.apache.org/plugins/maven-install-plugin/usage.html

- You need to know the hostname and JNDI port of the server hosting the session beans.

To invoke a session bean from a remote client you must first configure the project correctly.

Procedure 3.21. Add Maven Project Configuration for Remote Invocation of Session Beans

1. **Add the required project dependencies**
   The **pom.xml** for the project must be updated to include the necessary dependencies.

2. **Add the jboss-ejb-client.properties file**
   The JBoss EJB client API expects to find a file in the root of the project named **jboss-ejb-client.properties** that contains the connection information for the JNDI service. Add this file to the **src/main/resources/** directory of your project with the following content.

   ```
   # In the following line, set SSL_ENABLED to true for SSL
   remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLE
   D=false
   remote.connections=default
   # Uncomment the following line to set SSL_STARTTLS to true for SSL
   #
   remote.connection.default.connect.options.org.xnio.Options.SSL_START
   TLS=true
   remote.connection.default.host=localhost
   remote.connection.default.port = 4447
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOANONYMOUS=false
   # Add any of the following SASL options if required
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOANONYMOUS=false
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOPLAINTEXT=false
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_DISA
   LLOWED_MECHANISMS=JBOSS-LOCAL-USER
   ```

   Change the host name and port to match your server. **4447** is the default port number. For a secure connection, set the **SSL_ENABLED** line to **true** and uncomment the **SSL_STARTTLS** line. The Remoting interface in the container supports secured and unsecured connections using the same port.

3. **Add dependencies for the remote business interfaces**
   Add the Maven dependencies to the **pom.xml** for the remote business interfaces of the session beans.

   ```
   <dependency>
      <groupId>org.jboss.as.quickstarts</groupId>
      <artifactId>jboss-as-ejb-remote-server-side</artifactId>
      <type>ejb-client</type>
      <version>${project.version}</version>
   </dependency>
   ```

Now that the project has been configured correctly, you can add the code to access and invoke the session beans.

Procedure 3.22. Obtain a Bean Proxy using JNDI and Invoke Methods of the Bean

1. **Handle checked exceptions**
   Two of the methods used in the following code (**`InitialContext()`** and **`lookup()`**) have a checked exception of type **`javax.naming.NamingException`**. These method calls must either be enclosed in a try/catch block that catches **`NamingException`** or in a method that is declared to throw **`NamingException`**. The **`ejb-remote`** quickstart uses the second technique.

2. **Create a JNDI Context**
   A JNDI Context object provides the mechanism for requesting resources from the server. Create a JNDI context using the following code:

   ```
   final Hashtable jndiProperties = new Hashtable();
   jndiProperties.put(Context.URL_PKG_PREFIXES,
   "org.jboss.ejb.client.naming");
   final Context context = new InitialContext(jndiProperties);
   ```

   The connection properties for the JNDI service are read from the **`jboss-ejb-client.properties`** file.

3. **Use the JNDI Context's lookup() method to obtain a bean proxy**
   Invoke the **`lookup()`** method of the bean proxy and pass it the JNDI name of the session bean you require. This will return an object that must be cast to the type of the remote business interface that contains the methods you want to invoke.

   ```
   final RemoteCalculator statelessRemoteCalculator =
   (RemoteCalculator) context.lookup(
       "ejb:/jboss-as-ejb-remote-server-side/CalculatorBean!" +
       RemoteCalculator.class.getName());
   ```

   Session bean JNDI names are defined using a special syntax. For more information, see Section 3.2.10.3, “EJB JNDI Naming Reference” .

4. **Invoke methods**
   Now that you have a proxy bean object you can invoke any of the methods contained in the remote business interface.

   ```
   int a = 204;
   int b = 340;
   System.out.println("Adding " + a + " and " + b + " via the remote
   stateless calculator deployed on the server");
   int sum = statelessRemoteCalculator.add(a, b);
   System.out.println("Remote calculator returned sum = " + sum);
   ```

   The proxy bean passes the method invocation request to the session bean on the server, where it is executed. The result is returned to the proxy bean which then returns it to the caller. The communication between the proxy bean and the remote session bean is transparent to the caller.

You should now be able to configure a Maven project to support invoking session beans on a remote server and write the code invoke the session beans methods using a proxy bean retrieved from the server using JNDI.

Report a bug

3.2.10.3. EJB JNDI Naming Reference

The JNDI lookup name for a session bean has the syntax of:

```
 ejb:<appName>/<moduleName>/<distinctName>/<beanName>!<viewClassName>?
stateful
```

<appName>

If the session bean's JAR file has been deployed within an enterprise archive (EAR) then this is the name of that EAR. By default, the name of an EAR is its filename without the **.ear** suffix. The application name can also be overridden in its **application.xml** file. If the session bean is not deployed in an EAR then leave this blank.

<moduleName>

The module name is the name of the JAR file that the session bean is deployed in. By the default, the name of the JAR file is its filename without the **.jar** suffix. The module name can also be overridden in the JAR's **ejb-jar.xml** file.

<distinctName>

JBoss EAP 6 allows each deployment to specify an optional distinct name. If the deployment does not have a distinct name then leave this blank.

<beanName>

The bean name is the classname of the session bean to be invoked.

<viewClassName>

The view class name is the fully qualified classname of the remote interface. This includes the package name of the interface.

?stateful

The **?stateful** suffix is required when the JNDI name refers to a stateful session bean. It is not included for other bean types.

Report a bug

3.2.11. EJB 2.x Changes

3.2.11.1. Update Applications That Use EJB 2.x

JBoss EAP 6 provides support for EJB 2.x, however, you need to make a few code modifications and start the server with the full profile.

Procedure 3.23. Run EJB 2.x on JBoss EAP 6

1. **Modify the Code to Use the New JNDI Namespace Rules**
   As with EJB 3.0, you must use the full JNDI prefix with EJB 2.x. For more information on the new JNDI namespace rules and code examples, see Section 3.1.8.1, "Update Application JNDI Namespace Names".

   Examples showing how to update JNDI namespaces from previous releases can be found here: Section 3.1.8.5, "Examples of JNDI Namespaces in Previous Releases and How They are Specified in JBoss EAP 6".

2. **Modify the `jboss-web.xml` File Descriptor**
   Modify the **`<jndi-name>`** for each **`<ejb-ref>`** to use the new JNDI fully qualified lookup format.

3. **Replace the `jboss.xml` deployment descriptor file**
   The **`jboss-ejb3.xml`** deployment descriptor replaces the **`jboss.xml`** deployment descriptor to override and add to the features provided by the Java Enterprise Edition (EE) defined **`ejb-jar.xml`** deployment descriptor. The new file is incompatible with **`jboss.xml`**, and the **`jboss.xml`** is now ignored in deployments.

4. **Start the Server with the Full Profiles**
   EJB 2.x requires the Java Enterprise Edition 6 Full Profile. To start JBoss EAP 6 with the full profile, pass the argument **`-c standalone-full.xml`** on the command line when you start the server.

5. **Clustering is no longer supported**
   Clustering of EJB 2.x entity beans is no longer supported in JBoss EAP 6.

Report a bug

3.2.12. JBoss AOP Changes

3.2.12.1. Update Applications That Use JBoss AOP

JBoss AOP (Aspect Oriented Programming) is no longer included in JBoss EAP 6. In previous releases, JBoss AOP was used by the EJB container. In JBoss EAP 6, the EJB container uses a new mechanism. If your application uses JBoss AOP, you must modify your application code as follows.

Refactor the Application

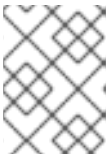
- Standard EJB3 configurations that were previously made in the **`ejb3-interceptors-aop.xml`** file are now configured in the server configuration file. For a standalone server, this is the **`standalone/configuration/standalone-full.xml`** file. If you are running your server in a managed domain, this is the **`domain/configuration/domain.xml`** file.

- Server side AOP Interceptors should be modified to use the standard Java EE **`Interceptor`**. For more information about container interceptors and how to use a client side interceptor in an application, refer to the chapter entitled *Container Interceptors* in the *Development Guide* for JBoss EAP 6 located on the Customer Portal at https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Use JBoss AOP Libraries

- If you are not able to refactor the code, you can obtain a copy of the JBoss AOP libraries and bundle them with the application. The AOP libraries may work in JBoss EAP 6, but are not deployed. You can manually deploy them by using the following command line argument when

you start your server: **Djboss.aop.path=*PATH_TO_AOP_CONFIG***



NOTE

Although the JBoss AOP libraries may work in JBoss EAP 6, this not a supported configuration.

Report a bug

3.2.13. Migrate Seam 2.2 Applications

3.2.13.1. Migrate Seam 2.2 Archives to JBoss EAP 6

Overview

When you migrate a Seam 2.2 application, you need to configure the datasource and specify any module dependencies. You also need to determine if the application has any dependencies on archives that do not ship with JBoss EAP 6 and copy any dependent JARs into the application **lib/** directory.

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

Procedure 3.24. Migrate Seam 2.2 Archives

1. **Update the datasource configuration**
   Some Seam 2.2 examples use the default JDBC datasource named **java:/ExampleDS**. This default datasource has changed in JBoss EAP 6 to **java:jboss/datasources/ExampleDS**. If your application uses the example database, you can do one of the following:

   - If you want to use the example database that ships with JBoss EAP 6, modify the **META-INF/persistence.xml** file to replace the existing **jta-data-source** element with the example database datasource JNDI name:

     ```
     <!-- <jta-data-source>java:/ExampleDS</jta-data-source> -->
     <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
     ```

   - If you prefer to keep your existing database, you can add the datasource definition to the ***EAP_HOME*/standalone/configuration/standalone.xml** file.

     IMPORTANT

     You must stop the server before editing the server configuration file for your change to be persisted on server restart.

     The following definition is a copy of the default HSQL datasource defined in JBoss EAP 6:

```
<datasource name="ExampleDS" jndi-name="java:/ExampleDS"
enabled="true" jta="true" use-java-context="true" use-ccm="true">
   <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1</connection-url>
   <driver>h2</driver>
   <security>
      <user-name>sa</user-name>
      <password>sa</password>
   </security>
</datasource>
```

- You can also add the datasource definition using the Management CLI command line interface. The following is an example of the syntax you must use to add a datasource. The "\" at the end of line indicates the continuation of the command on the following line.

Example 3.1. Example of syntax to add the datasource definition

```
$ EAP_HOME/bin/jboss-cli --connect
[standalone@localhost:9999 /] data-source add --name=ExampleDS
--jndi-name=java:/ExampleDS \
      --connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-1 --
driver-name=h2 \
      --user-name=sa --password=sa
```

For more information on how to configure a datasource, see Section 3.1.6.2, “Update the DataSource Configuration”.

2. **Add any required dependencies**
Because Seam 2.2 applications use JSF 1.2, you need to add dependencies for the JSF 1.2 modules and exclude the JSF 2.0 modules. To accomplish this, you need to create a **`jboss-deployment-structure.xml`** file in the EAR's **`META-INF/`** directory that contains the following data:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
            <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
            <module name="com.sun.jsf-impl" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
            <module name="com.sun.jsf-impl" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

If your application uses any third-party logging frameworks you need to add those dependencies as described here: Section 3.1.4.1, “Modify Logging Dependencies”.

3. **If your application uses Hibernate 3.x, first try to run the application using the Hibernate 4 libraries**
   If your application does not use the Seam Managed Persistence Context, Hibernate search, validation, or other features that have changed with Hibernate 4, you may be able to run with the Hibernate 4 libraries. However, if you see **`ClassNotFoundExceptions`** or **`ClassCastExceptions`** that point to Hibernate classes, or see errors similar to the following, you may have to follow the instructions in the next step and modify the application to use Hibernate 3.3 libraries.

   ```
   Caused by: java.lang.LinkageError: loader constraint
violation in interface itable initialization: when resolving method
"org.jboss.seam.persistence.HibernateSessionProxy.getSession(Lorg/hi
bernate/EntityMode;)Lorg/hibernate/Session;" the class loader
(instance of org/jboss/modules/ModuleClassLoader) of the current
class, org/jboss/seam/persistence/HibernateSessionProxy, and the
class loader (instance of org/jboss/modules/ModuleClassLoader) for
interface org/hibernate/Session have different Class objects for the
type org/hibernate/Session used in the signature
   ```

4. **Copy dependent archives from outside frameworks or other locations**
   If your application uses Hibernate 3.x and you are not able to use Hibernate 4 successfully with your application, you will need to copy the Hibernate 3.x JARs into the **`/lib`** directory and exclude the Hibernate module in the deployments section of the **`META-INF/jboss-deployment-structure.xml`** as follows:

   ```
   <jboss-deployment-structure xmlns="urn:jboss:deployment-
   structure:1.0">
    <deployment>
      <exclusions>
        <module name="org.hibernate"/>
      </exclusions>
    <deployment>
   </jboss-deployment-structure>
   ```

   There are additional steps you must take when you bundle Hibernate 3.x with your application. For more information, see Section 3.2.2.2, “Configure Changes for Applications That Use Hibernate and JPA”.

5. **Debug and resolve Seam 2.2 JNDI errors**
   When you migrate a Seam 2.2 application, you may see **`javax.naming.NameNotFoundException`** errors in the log like the following:

   ```
   javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not
   found in context ''
   ```

   If you don't want to modify JNDI lookups throughout the code, you can modify the application's **`components.xml`** file as follows:

   a. **Replace the existing core-init element**
      First, you need to replace the existing core-init element as follows:

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/>   -->
<core:init debug="true" distributable="false"/>
```

b. **Find the JNDI binding INFO messages in the server log**
Next, find the JNDI binding INFO messages that are printed in the server log when the application is deployed. The JNDI binding messages should look similar to this:

```
INFO
org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeployment
UnitProcessor (MSC service thread 1-1) JNDI bindings for session
bean
named AuthenticatorAction in deployment unit subdeployment
"jboss-seam-booking.jar" of deployment "jboss-seam-booking.ear"
are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:app/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator

java:module/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction
    java:app/jboss-seam-booking.jar/AuthenticatorAction
    java:module/AuthenticatorAction
```

c. **Add component elements**
For each JNDI binding INFO message in the log, add a matching **`component`** element to the **`components.xml`** file:

```
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking.jar/AuthenticatorAction" />
```

For more information on how to debug and resolve migration issues, see Section 4.2.1, “Debug and Resolve Migration Issues”.

For a list of known migration issues with Seam 2 archives, see Section 3.2.13.2, “Seam 2.2 Archive Migration Issues”.

Result

The Seam 2.2 archive deploys and runs successfully on JBoss EAP 6.

Report a bug

3.2.13.2. Seam 2.2 Archive Migration Issues

Seam 2.2 Drools and Java 7 are not compatible

Seam 2.2 Drools and Java 7 are incompatible and fail with an org.drools.RuntimeDroolsException: value '1.7' is not a valid language level error.

Seam 2.2.5 signed `cglib.jar` prevents the Spring example from working

When the Spring example is run using the signed **`cglib.jar`** that shipped with Seam 2.2.5 in JBoss EAP 5, it fails with the following root cause:

```
java.lang.SecurityException: class
"org.jboss.seam.example.spring.UserService$$EnhancerByCGLIB$$7d6c3d12"'s
signer information does not match signer information of other classes in
the same package
```

The work around for this issue is to unsign the **`cglib.jar`** as follows:

```
zip -d $SEAM_DIR/lib/cglib.jar META-INF/JBOSSCOD\*
```

Seambay example fails with `NotLoggedInException`

The cause of this issue is the SOAP message header being null when processing the message in the SOAPRequestHandler and consequently, the conversation ID not being set.

The work around for this issue is to override **`org.jboss.seam.webservice.SOAPRequestHandler.handleOutbound`**, as described in https://issues.jboss.org/browse/JBPAPP-8376.

Seambay example fails with `UnsupportedOperationException`: no transaction

This bug is caused by changes in the JNDI name of UserTransaction in JBoss EAP 6.

The work around for this issue is to override **`org.jboss.seam.transaction.Transaction.getUserTransaction`**, as described in https://issues.jboss.org/browse/JBPAPP-8322.

Tasks example throws `org.jboss.resteasy.spi.UnhandledException`: Unable to unmarshall request body

This bug is caused by the incompatibility between seam-resteasy-2.2.5 included in JBoss EAP 5.1.2) and RESTEasy 2.3.1.GA included in JBoss EAP 6.

The work around for this issue is to use the **`jboss-deployment-structure.xml`** to exclude resteasy-jaxrs, resteasy-jettison-provider, and resteasy-jaxb-provider from the main deployment and resteasy-jaxrs, resteasy-jettison-provider, resteasy-jaxb-provider, and resteasy-yaml-provider from the **`jboss-seam-tasks.war`** as described in https://issues.jboss.org/browse/JBPAPP-8315. It is then necessary to include the RESTEasy libraries bundled with Seam 2.2 in the EAR.

Deadlock between `org.jboss.seam.core.SynchronizationInterceptor` and stateful component instance EJB lock during an AJAX request

An error page with "Caused by javax.servlet.ServletException with message: "javax.el.ELException: /main.xhtml @36,71 value="#{hotelSearch.pageSize}": org.jboss.seam.core.LockTimeoutException: could not acquire lock on @Synchronized component: hotelSearch" or similar error message is displayed.

The problem is that Seam 2 does its own locking outside the stateful session bean (SFSB) lock and with a different scope. This means that if a thread accesses an EJB twice in the same transaction, after the first invocation it will have the SFSB lock, but not the seam lock. A second thread can then

acquire the seam lock, which will then hit the EJB lock and wait. When the first thread attempts its second invocation it will block on the seam 2 interceptor and deadlock. In Java EE 5, EJBs would throw an exception immediately on concurrent access. This behavior has changed in Java EE 6.

The work around for this issue is to add @AccessTimeout(0) to the EJB. This will cause it to throw a **`ConcurrentAccessException`** immediately when this situation occurs.

Dvdstore example create order fails with a `javax.ejb.EJBTransactionRolledbackException`

The dvdstore example displays the following error:

```
JBAS011437: Found extended persistence context in SFSB invocation call
stack but that cannot be used because the transaction already has a
transactional context associated with it. This can be avoided by
changing application code, either eliminate the extended persistence
context or the transactional context. See JPA spec 2.0 section 7.6.3.1.
```

This problem is due to changes in the JPA specification.

The fix for this issue is to change the persistence context to **`transactional`** in the **`CheckoutAction`** and **`ShowOrdersAction`** classes and use the entity manager merge operation in the **`cancelOrder`** and **`detailOrder`** methods.

JBoss Cache Seam Cache provider cannot be used in JBoss EAP 6

JBoss Cache is not supported in JBoss EAP 6. This causes JBoss Cache Seam Cache provider to fail in a Seam application on the application server with a

```
java.lang.NoClassDefFoundError: org/jboss/util/xml/JBossEntityResolver
```

.

Hibernate 3.3.x Auto scan for JPA entities issue with JBoss EAP 6

The fix for this issue is to list all the entity classes in the persistence.xml file manually. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
    <persistence-unit name="example_pu">
        <description>Hibernate 3 Persistence Unit.</description>
        <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
        <properties>
            <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
        </properties>
        <class>com.acme.Foo</class>
        <class>com.acme.Bar</class>
    </persistence-unit>
</persistence>
```

Calling EJB Seam components from non-EJB Threads results in a javax.naming.NameNotFoundException

This issue is a result of changes in JBoss EAP 6 to implement the new modular class loading system and to adopt the new standardized JNDI namespace conventions. The **`java:app`** namespace is designated for names shared by all components in a single application. Non-EE threads, such as Quartz asynchronous threads, must use the **`java:global`** namespace, which is shared by all applications deployed in an application server instance.

If you receive a **`javax.naming.NameNotFoundException`** when you try to call EJB Seam components from Quartz asynchronous methods, you must modify the **`components.xml`** file to use the global JNDI name, for example:

```
<component class="org.jboss.seam.example.quartz.MyBean" jndi-
name="java:global/seam-quartz/quartz-ejb/myBean"/>
```

For more information on JNDI changes, refer to the following topic: Section 3.1.8.1, “Update Application JNDI Namespace Names” . For more information on this specific issue, refer to *BZ#948215 - Seam2.3 javax.naming.NameNotFoundException trying to call EJB Seam components from quartz asynchronous methods* in the *2.2.0 Release Notes* for *JBoss Web Framework Kit* on the Customer Portal at https://access.redhat.com/site/documentation/JBoss_Web_Framework_Kit/.

Report a bug

3.2.14. Migrate Spring Applications

3.2.14.1. Migrate Spring Applications

Information about migrating Spring applications can be found in the *JBoss Web Framework Kit* documentation. You can download this documentation from the Customer Portal at https://access.redhat.com. Click the **Knowledge → Product Documentation**, find *Red Hat JBoss Enterprise Middleware*, then click the *JBoss Web Framework Kit* link.

Report a bug

3.2.15. Other Changes That Affect Migration

3.2.15.1. Become Familiar with Other Changes That May Affect Your Migration

The following is a list of other changes in JBoss EAP 6 that could impact your migration efforts.

- Section 3.2.15.2, “Change the Maven Plug-in Name”
- Section 3.2.15.3, “Modify Client Applications”

Report a bug

3.2.15.2. Change the Maven Plug-in Name

The **`jboss-maven-plugin`** has not been updated and does not work in JBoss EAP 6. You must now use **`org.jboss.as.plugins:jboss-as-maven-plugin`** to deploy to the correct directory.

Report a bug

3.2.15.3. Modify Client Applications

If you plan to migrate a client application that will connect to JBoss EAP 6, be aware that the name and location of the JAR that bundles the client libraries has changed. This JAR is now named **`jboss-client.jar`** and is located in the ***`EAP_HOME`*`/bin/client/`** directory. It replaces the ***`EAP_HOME`*`/client/jbossall-client.jar`** and contains all the dependencies required to connect to JBoss EAP 6 from a remote client.

Report a bug

CHAPTER 4. TOOLS AND TIPS

4.1. RESOURCES TO ASSIST WITH MIGRATION

4.1.1. Resources to Assist in Your Migration

The following is a list of resources that may be of help when migrating an application to JBoss EAP 6.

Tools

There are several tools that help automate some of the configuration changes. For more information, see: Section 4.1.2, “Become Familiar with Tools That Can Assist with the Migration”.

Debugging Tips

For a list of the most common causes and resolutions of issues and errors you may see when you migrate your application, see: Section 4.2.1, “Debug and Resolve Migration Issues”.

Example migrations

For examples of applications that have been migrated to JBoss EAP 6, see: Section 4.3.1, “Review Migration of Example Applications”.

Report a bug

4.1.2. Become Familiar with Tools That Can Assist with the Migration

Summary

There are some tools that can assist you in your migration efforts. The following is a list of these tools along with a description of what they do.

Tattletale

With the change to modular class loading, you need to find and rectify application dependencies. Tattletale can help you identify dependent module names and generate the configuration XML for your application.

Section 4.1.3, “Use Tattletale to Find Application Dependencies”

IronJacamar Migration Tool

In JBoss EAP 6, datasources and resource adapters are no longer configured in a separate file. They are now defined in the server configuration file and use new schemas. The IronJacamar Migration Tool can help convert the old configuration into the format expected by JBoss EAP 6.

Section 4.1.6, “Use the IronJacamar Tool to Migrate Datasource and Resource Adapter Configurations”

Report a bug

4.1.3. Use Tattletale to Find Application Dependencies

Summary

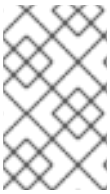
Due to the modular class loading changes in JBoss EAP 6, you might see **`ClassNotFoundException`**

or **ClassCastException** traces in the JBoss log when you migrate your application. To resolve these errors, you need to find the JARs that contain the classes specified by the exceptions.

Tattletale is an excellent 3rd party tool that recursively scans your application and provides detailed reports about its contents. Tattletale 1.2.0.Beta2 or later contains additional support to help with the new JBoss Modules class loading used in JBoss EAP 6. Tattletale's "JBoss AS 7" report can be used to automatically identify and generate dependent module names to include your application's **jboss-deployment-structure.xml** file.

Procedure 4.1. Install and run Tattletale to find application dependencies

1. Section 4.1.4, “Download and Install Tattletale”
2. Section 4.1.5, “Create and Review the Tattletale Report”



NOTE

Tattletale is a 3rd party tool and not supported as part of JBoss EAP 6. For the most current documentation on how to install and use Tattletale, go to the Tattletale web site at http://www.jboss.org/tattletale.

Report a bug

4.1.4. Download and Install Tattletale

Procedure 4.2. Download and Install Tattletale

1. Download Tattletale version 1.2.0.Beta2 or newer from http://sourceforge.net/projects/jboss/files/JBoss%20Tattletale.
2. Unzip the file into the directory of your choice.
3. Modify the **TATTLETALE_HOME/jboss-tattletale.properties** file by doing the following:
   a. Add **ee6** and **as7** to the **profiles** property.

   ```
   profiles=java5, java6, ee6, as7
   ```

   b. Uncomment the **scan** and **reports** properties.

Report a bug

4.1.5. Create and Review the Tattletale Report

1. Create the Tattletale report by issuing the command: **java -jar *TATTLETALE_HOME*/tattletale.jar *APPLICATION_ARCHIVE* *OUTPUT_DIRECTORY***

   For example: **java -jar tattletale-1.2.0.Beta2/tattletale.jar applications/jboss-seam-booking.ear output-results/**

2. In a browser, open the ***OUTPUT_DIRECTORY*/index.html** file and click on "JBoss AS 7" under the "Reports" section.

a. The column on the left lists the archives used by the application. Click on the *ARCHIVE_NAME* link to view details about the archive, such as its location, manifest information, and classes it contains.

b. The **`jboss-deployment-structure.xml`** link in the column on the right shows how to specify the module dependency for the archive named in the left column. Click on this link to see how to define the deployment dependency module information for this archive.

Report a bug

4.1.6. Use the IronJacamar Tool to Migrate Datasource and Resource Adapter Configurations

Summary

In previous versions of the application server, datasources and resource adapters were configured and deployed using a file with a suffix of **`*-ds.xml`**. The IronJacamar 1.1 distribution contains a migration tool that can be used to convert these configuration files into the format expected by JBoss EAP 6. The tool parses the source configuration file from the previous release, then creates and writes the XML configuration to an output file in the new format. This XML can then be copied and pasted under the correct subsystem in the JBoss EAP 6 server configuration file. This tool makes a best effort to convert old attributes and elements into the new format, however, it may be necessary to make additional modifications to the generated file.

Procedure 4.3. Install and run the IronJacamar Migration tool

1. Section 4.1.7, “Download and Install the IronJacamar Migration Tool”
2. Section 4.1.8, “Use the IronJacamar Migration Tool to Convert a Datasource Configuration File”
3. Section 4.1.9, “Use the IronJacamar Migration Tool to Convert a Resource Adapter Configuration File”

NOTE

The IronJacamar Migration tool is a 3rd party tool and not supported as part of JBoss EAP 6. For more information about IronJacamar, go to http://www.ironjacamar.org/. For the most current documentation on how to install and use this tool, go to http://www.ironjacamar.org/documentation.html.

Report a bug

4.1.7. Download and Install the IronJacamar Migration Tool

NOTE

The migration tool is only available in IronJacamar 1.1 version or higher.

1. Download the IronJacamar 1.1 or greater distribution from here: http://www.ironjacamar.org/download.html/
2. Unzip the downloaded file into a directory of your choice.
3. Find the converter script in the IronJacamar distribution.

- The Linux script is located here: ***IRONJACAMAR_HOME*/doc/as/converter.sh**
- The Windows batch file is located here: ***IRONJACAMAR_HOME*/doc/as/converter.bat**

Report a bug

4.1.8. Use the IronJacamar Migration Tool to Convert a Datasource Configuration File

Procedure 4.4. Convert a Datasource Configuration File

1. Open a command line and navigate to the ***IRONJACAMAR_HOME*/docs/as/** directory.
2. Run the converter script by typing the following command:
   - For Linux: **./converter.sh -ds *SOURCE_FILE TARGET_FILE***
   - For Microsoft Windows: **./converter.bat -ds *SOURCE_FILE TARGET_FILE***

   The ***SOURCE_FILE*** is the datasource -ds.xml file from the previous release. The ***TARGET_FILE*** contains the new configuration.

   For example, to convert the **jboss-seam-booking-ds.xml** datasource configuration file located in the current directory, you would type:
   - For Linux: **./converter.sh -ds jboss-seam-booking-ds.xmlnew-datasource-config.xml**
   - For Microsoft Windows: **./converter.bat -ds jboss-seam-booking-ds.xmlnew-datasource-config.xml**

   Note that the parameter for datasource conversion is **-ds**.
3. Copy the **<datasource>** element from the target file and paste it into the server configuration file under the **<subsystem xmlns="urn:jboss:domain:datasources:1.1"><datasources>** element.

   IMPORTANT

   You must stop the server before editing the server configuration file for your change to be persisted on server restart.

   - If you are running in a managed domain, copy the XML into the ***EAP_HOME*/domain/configuration/domain.xml** file.
   - If you are running as a standalone server, copy the XML into the ***EAP_HOME*/standalone/configuration/standalone.xml** file.
4. Modify the generated XML in the new configuration file.

   Here is an example of the **jboss-seam-booking-ds.xml** datasource configuration file for the Seam 2.2 Booking example that shipped with JBoss EAP 5.x:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<datasources>
  <local-tx-datasource>
    <jndi-name>bookingDatasource</jndi-name>
    <connection-url>jdbc:hsqldb:.</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

The following is the configuration file that was generated by running the converter script. The generated file contains a **<driver-class>** element. The preferred way to define the driver class in JBoss EAP 6 is to use a **<driver>** element. Here is the resulting XML in the JBoss EAP 6 configuration file with modifications to comment out the **<driver-class>** element and add the corresponding **<driver>** element:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
  <datasources>
    <datasource enabled="true" jndi-
name="java:jboss/datasources/bookingDatasource" jta="true"
            pool-name="bookingDatasource" use-ccm="true" use-java-
context="true">
      <connection-url>jdbc:hsqldb:.</connection-url>
      <!-- Comment out the following driver-class element
           since it is not the preferred way to define this.
           <driver-class>org.hsqldb.jdbcDriver</driver-class>     -
->
      <transaction-isolation>TRANSACTION_NONE</transaction-
isolation>
      <pool>
        <prefill>false</prefill>
        <use-strict-min>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
      </pool>
      <security>
        <user-name>sa</user-name>
        <password/>
      </security>
      <validation>
        <validate-on-match>false</validate-on-match>
        <background-validation>false</background-validation>
        <use-fast-fail>false</use-fast-fail>
      </validation>
      <timeout/>
      <statement>
        <track-statements>false</track-statements>
      </statement>
    </datasource>
    <drivers>
      <!-- The following driver element was not in the
     XML target file. It was created manually. -->
      <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
        </driver>
      </drivers>
```

```
  </datasources>
</subsystem>
```

Report a bug

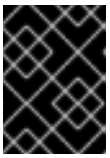
4.1.9. Use the IronJacamar Migration Tool to Convert a Resource Adapter Configuration File

1. Open a command line and navigate to the ***IRONJACAMAR_HOME*/docs/as/** directory.
2. Run the converter script by typing the following command:
   - For Linux: **./converter.sh -ra *SOURCE_FILE TARGET_FILE***
   - For Microsoft Windows: **./converter.bat -ra *SOURCE_FILE TARGET_FILE***

   The ***SOURCE_FILE*** is the resource adapter -ds.xml file from the previous release. The ***TARGET_FILE*** contains the new configuration.

   For example, to convert the **mttestadapter-ds.xml** resource adapter configuration file located in the current directory, you would type:
   - For Linux: **./converter.sh -ra mttestadapter-ds.xmlnew-adapter-config.xml**
   - For Microsoft Windows: **./converter.bat -ra mttestadapter-ds.xmlnew-adapter-config.xml**

   Note that the parameter for resource adapter conversion is **-ra**.
3. Copy the entire **<resource-adapters>** element from the target file and paste it into the server configuration file under the **<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">** element.

   IMPORTANT

   You must stop the server before editing the server configuration file for your change to be persisted on server restart.

   - If you are running in a managed domain, copy the XML into the ***EAP_HOME*/domain/configuration/domain.xml** file.
   - If you are running as a standalone server,copy the XML into the ***EAP_HOME*/standalone/configuration/standalone.xml** file.
4. Modify the generated XML in the new configuration file.

   Here is an example of the **mttestadapter-ds.xml** resource adapter configuration file from the JBoss EAP 5.x TestSuite:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--
=========================================================================
```

```
-->
  <!-- ConnectionManager setup for jboss test adapter
-->
  <!-- Build jmx-api (build/build.sh all) and view for config
documentation -->
  <!--
=========================================================================
-->
<connection-factories>
  <tx-connection-factory>
    <jndi-name>JBossTestCF</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCF2</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCFByTx</jndi-name>
    <xa-transaction/>
    <track-connection-by-tx>true</track-connection-by-tx>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
```

```
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
</connection-factories>
```

The following is the configuration file that was generated by running the converter script. Replace the class-name attribute value "FIXME_MCF_CLASS_NAME" in the generated XML with the correct class name of the managed connection factory, in this case, "org.jboss.test.jca.adapter.TestManagedConnectionFactory". Here is the resulting XML in the JBoss EAP 6 configuration file with modifications to the **<class-name>** element value:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
  <resource-adapters>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
  <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
  <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
    jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
    use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
      <prefill>false</prefill>
      <use-strict-min>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
```

```
    <validation>
      <background-validation>false</background-validation>
      <use-fast-fail>false</use-fast-fail>
    </validation>
  </connection-definition>
      </connection-definitions>
    </resource-adapter>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
  <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
   <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
      <prefill>false</prefill>
      <use-strict-min>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>false</background-validation>
      <use-fast-fail>false</use-fast-fail>
    </validation>
  </connection-definition>
      </connection-definitions>
    </resource-adapter>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
  <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
  <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
     jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
```

```
     use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
      <prefill>false</prefill>
      <use-strict-min>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>false</background-validation>
      <use-fast-fail>false</use-fast-fail>
    </validation>
  </connection-definition>
      </connection-definitions>
    </resource-adapter>
  </resource-adapters>
</subsystem>
```

Report a bug

4.2. DEBUG MIGRATION ISSUES

4.2.1. Debug and Resolve Migration Issues

Due to class loading, JNDI naming rules, and other changes in the application server, you may encounter exceptions or other errors if you try to deploy your application "as-is". The following describes how to resolve some of the more common exceptions and errors you might encounter.

- Section 4.2.2, “Debug and Resolve ClassNotFoundExceptions and NoClassDefFoundErrors”
- Section 4.2.5, “Debug and Resolve ClassCastExceptions”
- Section 4.2.6, “Debug and Resolve DuplicateServiceExceptions”
- Section 4.2.7, “Debug and Resolve JBoss Seam Debug Page Errors”

Report a bug

4.2.2. Debug and Resolve ClassNotFoundExceptions and NoClassDefFoundErrors

Summary

ClassNotFoundExceptions usually occur due to an unresolved dependency. This means you must explicitly define the dependencies on other modules or copy JARs from external sources.

1. First, try to find the missing dependency. This is described in more detail here: Section 4.2.3, “Find the JBoss Module Dependency”
2. If there is not a module for the missing class, find the JAR in the previous install. For more information, see Section 4.2.4, “Find the JAR in the Previous Install”

Report a bug

4.2.3. Find the JBoss Module Dependency

To resolve the dependency, first, try to find the module that contains the class specified by the **`ClassNotFoundException`** by looking in the ***`EAP_HOME`*`/modules/system/layers/base/`** directory. If you find a module for the class, you must add a dependency to the manifest entry.

For example, if you see this ClassNotFoundException trace in the log:

```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log
    from [Module "deployment.TopicIndex.war:main" from Service Module
Loader]
    at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:188)
```

Find the JBoss module containing this class by doing the following:

Procedure 4.5. Find the Dependency

1. First determine if there is an obvious module for the class.
   a. Navigate to the ***`EAP_HOME`*`/modules/system/layers/base/`** directory and look for the module path matching class named in the **`ClassNotFoundException`**.

      You find the module path **`org/apache/commons/logging/`**.

   b. Open the ***`EAP_HOME`*`/modules/system/layers/base/org/apache/commons/logging/main/module.xml`** file and find the module name. In this case, it is "org.apache.commons.logging".

   c. Add the module name to the Dependencies in the **`MANIFEST.MF`** file:

      ```
      Manifest-Version: 1.0
      Dependencies: org.apache.commons.logging
      ```

2. If there is no obvious module path for the class, you may need to find the dependency in another location.
   a. Find the class named by the **`ClassNotFoundException`** in the Tattletale report.

b. Find the module containing the JAR in the ***EAP_HOME*/modules** directory and find the module name as in the previous step.

Report a bug

4.2.4. Find the JAR in the Previous Install

If the class is not found in a JAR packaged in a module defined by the server, find the JAR in your *EAP5_HOME* install or your prior server's **lib/** directory.

For example, if you see this **ClassNotFoundException** trace in the log:

```
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/validator/ClassValidator at
java.lang.Class.getDeclaredMethods0(Native Method)
```

Find the JAR containing this class by doing the following:

1. Open a terminal and navigate to the ***EAP5_HOME*/** directory.
2. Issue the command:

   ```
   grep 'org.hibernate.validator.ClassValidator' `find . \-name '*.jar'`
   ```

3. You might see more than one result. In this case, the following result is the JAR we need:

   ```
   Binary file ./jboss-eap-5.1/seam/lib/hibernate-validator.jar matches
   ```

4. Copy this JAR to the application's **lib/** directory.

   If you find that you need a large number of JARs, it may be easier to define a module for the classes. For more information, refer to *Modules* in the chapter entitled *Get Started Developing Applications* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

5. Rebuild and redeploy the application.

Report a bug

4.2.5. Debug and Resolve ClassCastExceptions

ClassCastExceptions often happen because a class is being loaded by a different class loader than the class it extends. They can also be a result of the same class existing in multiple JARs.

1. Search the application to find all JAR(s) that contain the class named by the **ClassCastException**. If there is a module defined for the class, find and remove the duplicate JAR(s) from the application's WAR or EAR.
2. Find the JBoss module containing the class and explicitly define the dependency in the **MANIFEST.MF** file or in the **jboss-deployment-structure.xml** file. For more information, refer to *Class Loading and Subdeployments* in the chapter entitled *Class Loading and Modules* in the *Development Guide* for JBoss EAP 6 on https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

3. If you are not able to resolve it using the steps above, you can often determine the cause of the problem by printing the class loader information to the log. For example, you see the following **`ClassCastException`** in the log:

```
java.lang.ClassCastException: com.example1.CustomClass1 cannot be
cast to com.example2.CustomClass2
```

   a. In your code, print the class loader information for the classes named by the **`ClassCastException`** to the log, for example:

```
logger.info("Class loader for CustomClass1: " +

com.example1.CustomClass1.getClass().getClassLoader().toString())
;
logger.info("Class loader for CustomClass2: " +

com.example2.CustomClass2.getClass().getClassLoader().toString())
;
```

   b. The information in the log shows which modules are loading the classes and, based on your application, you need to remove or move the conflicting JAR(s).

Report a bug

4.2.6. Debug and Resolve DuplicateServiceExceptions

If you get a DuplicateServiceException for a subdeployment of a JAR or a message that the WAR application has already been installed when you deploy your EAR in JBoss EAP 6, it may be due to changes in the way JBossWS handles the deployment.

The JBossWS 3.3.0 release introduced a new Context Root Mapping Algorithm for servlet based endpoints to allow it to become seamlessly compatible with TCK6. If the application EAR archive contains a WAR and a JAR with the same name, JBossWS may create a WAR context and web context with the same name. The web context conflicts with the WAR context and this results in deployment errors. Resolve the deployment issues in one of the following ways:

- Rename the JAR file to a name that is different than the WAR so the generated web and WAR contexts is unique.
- Provide a **`<context-root>`** element in the **`jboss-web.xml`** file.
- Provide a **`<context-root>`** element in the **`jboss-webservices.xml`** file.
- Customize the **`<context-root>`** element for the WAR in the **`application.xml`** file.

Report a bug

4.2.7. Debug and Resolve JBoss Seam Debug Page Errors

After you migrate and successfully deploy your application, you may encounter a runtime error that redirects you to the "JBoss Seam Debug" page. The URL for this page is "http://localhost:8080/*APPLICATION_CONTEXT*/debug.seam". This page allows you to view and inspect the Seam components in any of the Seam contexts associated with your current login session.

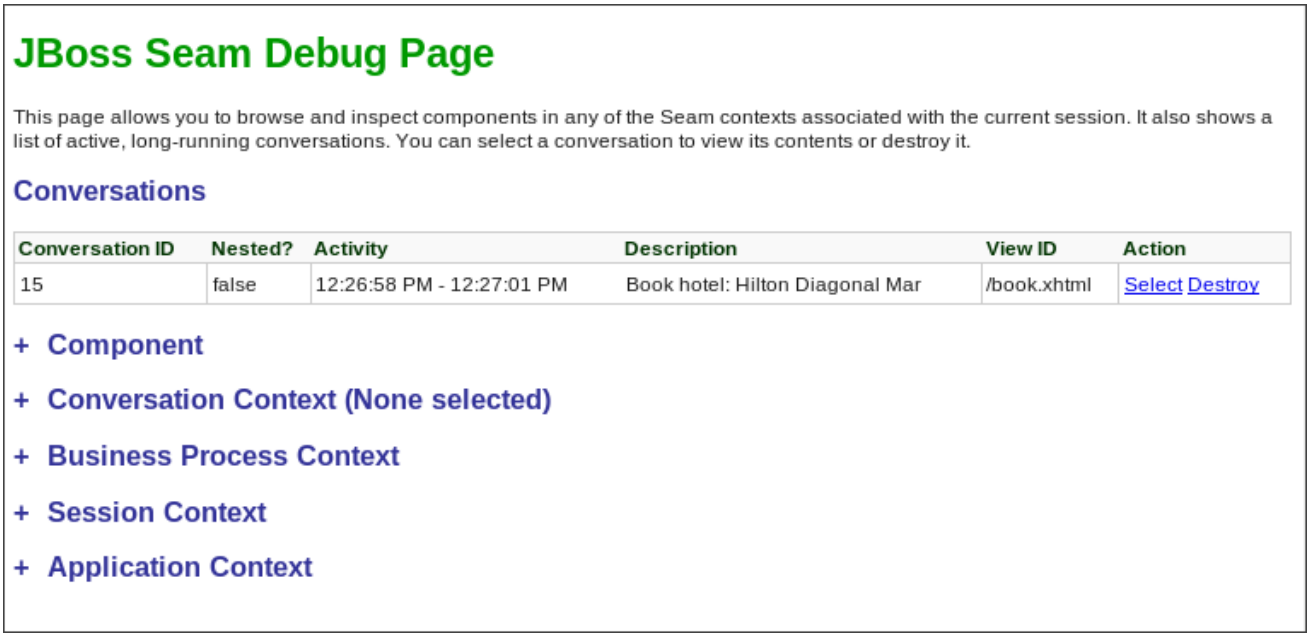


Figure 4.1. JBoss Seam Debug Page

The most likely reason you are redirected to this page is because Seam has caught an Exception that was not handled in the application code. The root cause of the exception can often be found in one of the links on the "JBoss Seam Debug Page".

1. Expand the **`Component`** section on the page and look for the **`org.jboss.seam.caughtException`** component.

2. The cause and stack trace should point you to the missing dependencies.

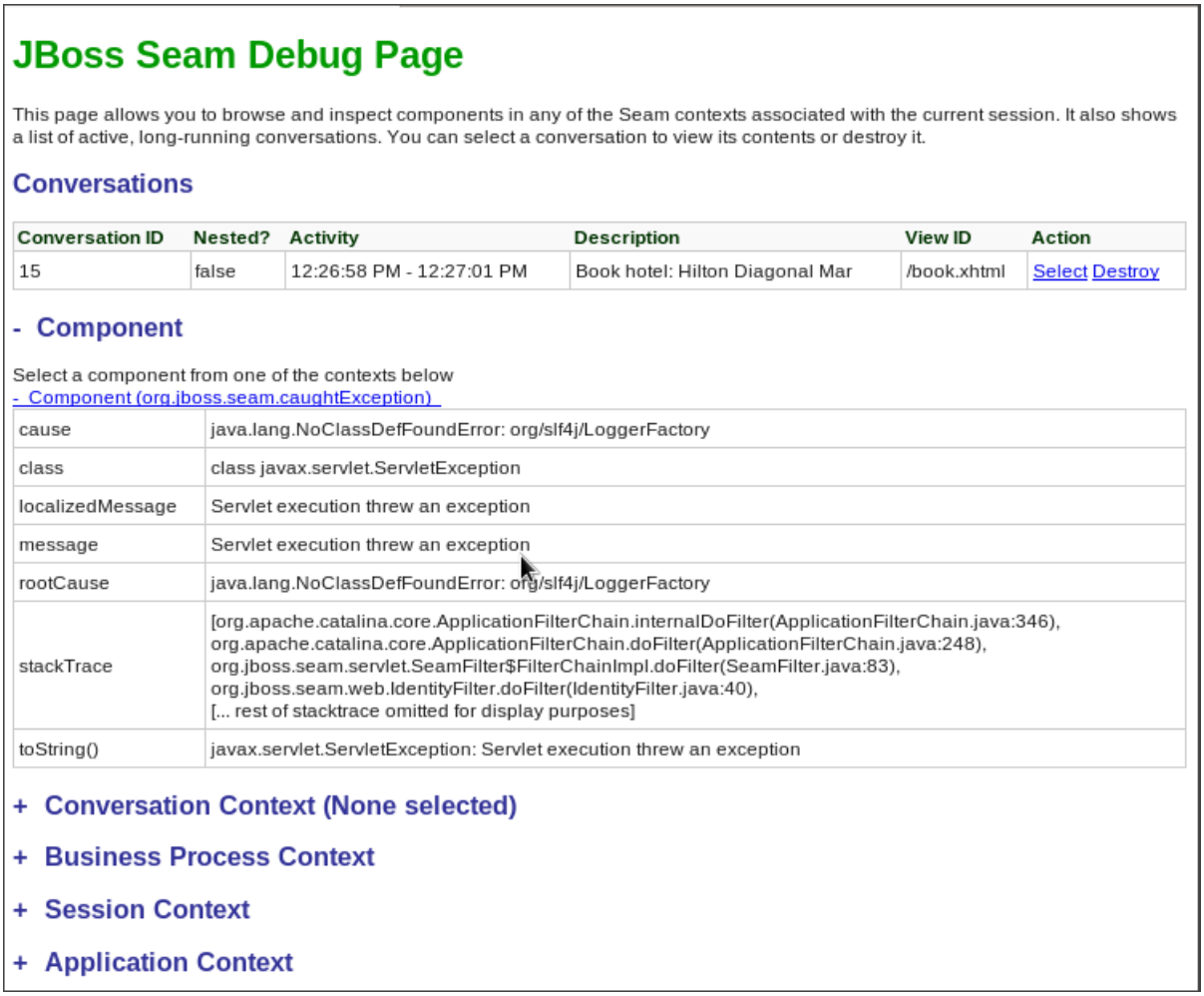


Figure 4.2. Component `org.jboss.seam.caughtException` information

3. Use the technique described in Section 4.2.2, “Debug and Resolve ClassNotFoundExceptions and NoClassDefFoundErrors” to resolve module dependencies.

   In the example above, the simplest solution is to add **`org.slf4j`** to the **`MANIFEST.MF`**

   ```
   Manifest-Version: 1.0
   Dependencies: org.slf4j
   ```

   Another option is to add a dependency for the module to the **`jboss-deployment-structure.xml`** file:

   ```
   <jboss-deployment-structure>
      <deployment>
           <dependencies>
             <module name="org.slf4j" />
           </dependencies>
       </deployment>
   </jboss-deployment-structure>
   ```

Report a bug

4.3. REVIEW MIGRATION OF EXAMPLE APPLICATIONS

4.3.1. Review Migration of Example Applications

Overview

The following is a list of JBoss EAP 5.x example applications that have been migrated to JBoss EAP 6. To view the details of what was changed in a particular application, click on the link below.

- Section 4.3.2, “Migrate the Seam 2.2 JPA Example to JBoss EAP 6”
- Section 4.3.3, “Migrate the Seam 2.2 Booking Example to JBoss EAP 6”
- Section 4.3.4, “Migrate the Seam 2.2 Booking Archive to JBoss EAP 6: Step-By-Step Instructions”

Report a bug

4.3.2. Migrate the Seam 2.2 JPA Example to JBoss EAP 6

Summary

The following task list summarizes the changes needed to successfully migrate the Seam 2.2 JPA example application to JBoss EAP 6. This example application can be found in the JBoss EAP 5.1 distribution under ***`EAP5.1_HOME`*`/jboss-eap-5.1/seam/examples/jpa/`**

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

Procedure 4.6. Migrate the Seam 2.2 JPA Example

1. **Remove the jboss-web.xml file**
   Remove the **`jboss-web.xml`** file from the **`jboss-seam-jpa.war/WEB-INF/`** directory. The class loading defined in the **`jboss-web.xml`** is now the default behavior.

2. **Modify the `jboss-seam-jpa.jar/META-INF/persistence.xml` file as follows.**

   a. Remove or comment out the **`hibernate.cache.provider_class`** property in the **`jboss-seam-jpa.war/WEB-INF/classes/META-INF/persistence.xml`** file:

   ```
   <!-- <property name="hibernate.cache.provider_class"
   value="org.hibernate.cache.HashtableCacheProvider"/> -->
   ```

   b. Add the provider module property to the **`jboss-seam-booking.jar/META-INF/persistence.xml`** file:

   ```
   <property name="jboss.as.jpa.providerModule" value="hibernate3-
   bundled" />
   ```

   c. Change the **`jta-data-source`** property to use the default JDBC datasource JNDI name:

```
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

3. **Add Seam 2.2 dependencies**
   Copy the following JARs from the Seam 2.2 distribution library, **`SEAM_HOME/lib/`**, into the **`jboss-seam-jpa.war/WEB-INF/lib/`** directory:
   - antlr.jar
   - slf4j-api.jar
   - slf4j-log4j12.jar
   - hibernate-entitymanager.jar
   - hibernate-core.jar
   - hibernate-annotations.jar
   - hibernate-commons-annotations.jar
   - hibernate-validator.jar
4. **Create a jboss-deployment-structure file to add remaining dependencies**
   Create a **`jboss-deployment-structure.xml`** file in the **`jboss-seam-jpa.war/WEB-INF/`** folder containing the following data:

```
<jboss-deployment-structure>
   <deployment>
        <exclusions>
          <module name="javax.faces.api" slot="main"/>
          <module name="com.sun.jsf-impl" slot="main"/>
          <module name="org.hibernate" slot="main"/>
        </exclusions>
        <dependencies>
          <module name="org.apache.log4j" />
          <module name="org.dom4j" />
          <module name="org.apache.commons.logging" />
          <module name="org.apache.commons.collections" />
          <module name="javax.faces.api" slot="1.2"/>
          <module name="com.sun.jsf-impl" slot="1.2"/>
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

Result:

The Seam 2.2 JPA example application deploys and runs successfully on JBoss EAP 6.

Report a bug

4.3.3. Migrate the Seam 2.2 Booking Example to JBoss EAP 6

Summary

The Seam 2.2 Booking EAR migration is more complicated than the Seam 2.2 JPA WAR example.

Documentation for the Seam 2.2 JPA WAR example migration can be found here: Section 4.3.2, “Migrate the Seam 2.2 JPA Example to JBoss EAP 6”. To migrate the application, you must do the following:

1. Initialize JSF 1.2 instead of the default JSF 2.
2. Bundle older versions of the Hibernate JARs rather than use those that ship with JBoss EAP 6.
3. Change the JNDI bindings to use the new Java EE 6 JNDI portable syntax.

The first 2 steps above were done in the Seam 2.2 JPA WAR example migration. The third step is new and is necessary because the EAR contains EJBs.

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

Procedure 4.7. Migrate the Seam 2.2 Booking example

1. **Create the `jboss-deployment-structure.xml` file**
   Create a new file named **`jboss-deployment-structure.xml`** in the **`jboss-seam-booking.ear/META-INF/`** and add the following content:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
        <dependencies>
          <module name="javax.faces.api" slot="1.2" export="true"/>
          <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
          <module name="org.apache.log4j" export="true"/>
          <module name="org.dom4j" export="true"/>
          <module name="org.apache.commons.logging" export="true"/>
          <module name="org.apache.commons.collections"
export="true"/>
        </dependencies>
        <exclusions>
          <module name="org.hibernate" slot="main"/>
       </exclusions>

  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
      <exclusions>
          <module name="javax.faces.api" slot="main"/>
          <module name="com.sun.jsf-impl" slot="main"/>
        </exclusions>
        <dependencies>
          <module name="javax.faces.api" slot="1.2"/>
          <module name="com.sun.jsf-impl" slot="1.2"/>
```

```
        </dependencies>
    </sub-deployment>
</jboss-deployment-structure>
```

2. **Modify the `jboss-seam-booking.jar/META-INF/persistence.xml` file as follows.**

   a. Remove or comment out the hibernate property for the cache provider class:

   ```
   <!-- <property name="hibernate.cache.provider_class"
   value="org.hibernate.cache.HashtableCacheProvider"/> -->
   ```

   b. Add the provider module property to the **`jboss-seam-booking.jar/META-INF/persistence.xml`** file:

   ```
   <property name="jboss.as.jpa.providerModule" value="hibernate3-
   bundled" />
   ```

   c. Change the **`jta-data-source`** property to use the default JDBC datasource JNDI name:

   ```
   <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
   source>
   ```

3. **Copy JARs from the Seam 2.2 distribution**
   Copy the following JARs from the Seam 2.2 distribution ***`EAP5.x_HOME`*`/jboss-eap5.x/seam/lib/`** into the **`jboss-seam-booking.ear/lib`** directory.

   ```
   antlr.jar
   slf4j-api.jar
   slf4j-log4j12.jar
   hibernate-core.jar
   hibernate-entitymanager.jar
   hibernate-validator.jar
   hibernate-annotations.jar
   hibernate-commons-annotations.jar
   ```

4. **Change the JNDI lookup names**
   Change JNDI lookup strings in the **`jboss-seam-booking.war/WEB-INF/components.xml`** file. Because of new JNDI portable rules, JBoss EAP 6 now binds EJBs using JNDI portable syntax rules and you cannot use the single jndiPattern that was used in JBoss EAP 5. This is what the application EJB JNDI lookup strings must be changed to JBoss EAP 6:

   ```
   java:global/jboss-seam-booking/jboss-seam-
   booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
   rching
   java:app/jboss-seam-
   booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
   rching
   java:module/HotelSearchingAction!org.jboss.seam.example.booking.Hote
   lSearching
   java:global/jboss-seam-booking/jboss-seam-
   booking/HotelSearchingAction
   java:app/jboss-seam-booking/HotelSearchingAction
   java:module/HotelSearchingAction
   ```

The JNDI lookup strings for the Seam 2.2 framework EJBs must be changed as follows:

```
java:global/jboss-seam-booking/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:app/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:module/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbS
ynchronizations
java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations
java:app/jboss-seam/EjbSynchronizations
java:module/EjbSynchronizations
```

You can take either of the following approaches:

a. **Add component elements**
You can add a **`jndi-name`** for every EJB to the **`WEB-INF/components.xml`**:

```
    <component
class="org.jboss.seam.transaction.EjbSynchronizations" jndi-
name="java:app/jboss-seam/EjbSynchronizations"/>
    <component
class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
    <component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking/AuthenticatorAction" />
    <component
class="org.jboss.seam.example.booking.BookingListAction"  jndi-
name="java:app/jboss-seam-booking/BookingListAction" />
    <component
class="org.jboss.seam.example.booking.RegisterAction" jndi-
name="java:app/jboss-seam-booking/RegisterAction" />
    <component
class="org.jboss.seam.example.booking.HotelSearchingAction" jndi-
name="java:app/jboss-seam-booking/HotelSearchingAction" />
    <component
class="org.jboss.seam.example.booking.HotelBookingAction" jndi-
name="java:app/jboss-seam-booking/HotelBookingAction" />
    <component
class="org.jboss.seam.example.booking.ChangePasswordAction" jndi-
name="java:app/jboss-seam-booking/ChangePasswordAction" />
```

b. You can modify the code by adding the **`@JNDIName(value="")`** annotation specifying the JNDI path. An example of the changed stateless session bean code is below. A detailed description of this process can be found in the Seam 2.2 reference documentation.

```
@Stateless
@Name("authenticator")
@JndiName(value="java:app/jboss-seam-
booking/AuthenticatorAction")
public class AuthenticatorAction
    implements Authenticator
```

```
{
...
}
```

Result:

The Seam 2.2 Booking application deploys and runs successfully on JBoss EAP 6.

Report a bug

4.3.4. Migrate the Seam 2.2 Booking Archive to JBoss EAP 6: Step-By-Step Instructions

This is a step-by-step guide on how to port the Seam 2.2 Booking application archive from JBoss EAP 5.1 to JBoss EAP 6. Although there are better approaches for migrating applications, many developers might be tempted to deploy the application archive as-is to the JBoss EAP 6 server to see what happens. The purpose of this document is to show the types of issues you might encounter when you do that and how you can debug and resolve those issues.

For this example, the application EAR is deployed to the ***EAP6_HOME*/standalone/deployments** directory with no changes other than extracting the archives. This allows you to easily modify the XML files contained within the archives as you encounter and resolve issues.

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

Procedure 4.8. Migrate the application

1. Section 4.3.5, “Build and Deploy the JBoss EAP 5.1 Version of the Seam 2.2 Booking Application”
2. Section 4.3.6, “Debug and Resolve Seam 2.2 Booking Archive Deployment Errors and Exceptions”
3. Section 4.3.7, “Debug and Resolve Seam 2.2 Booking Archive Runtime Errors and Exceptions”

At this point you are able to successfully access the application in a browser using the URL http://localhost:8080/seam-booking/. Login with demo/demo and you see the Booking welcome page.

Review the summary of changes

Section 4.3.8, “Review a Summary of the Changes Made When Migrating the Seam 2.2 Booking Application”

Report a bug

4.3.5. Build and Deploy the JBoss EAP 5.1 Version of the Seam 2.2 Booking Application

Before migrating this application, you need to build the JBoss EAP 5.1 Seam 2.2 Booking application, extract the archive, and copy it into the JBoss EAP 6 deployment folder.

Procedure 4.9. Build and deploy the EAR

1. Build the EAR:

```
$ cd /EAP5_HOME/jboss-eap5.1/seam/examples/booking
$ ANT_HOME/ant explode
```

2. Copy the EAR to the *EAP6_HOME* deployments directory:

```
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.ear EAP6_HOME/standalone/deployments/
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.war EAP6_HOME/standalone/deployments/jboss-seam.ear
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.jar EAP6_HOME/standalone/deployments/jboss-seam.ear
```

3. Start the JBoss EAP 6 server and check the log. You see:

```
INFO [org.jboss.as.deployment] (DeploymentScanner-threads - 1) Found
jboss-seam-booking.ear in deployment directory.
    To trigger deployment create a file called jboss-seam-
booking.ear.dodeploy
```

4. Create an empty file with the name **`jboss-seam-booking.ear.dodeploy`** and copy it into the ***`EAP6_HOME`*`/standalone/deployments`** directory. You need to copy this file into the deployments directory many times while migrating this application, so keep it in a location where you can easily find it. In the log, you should now see the following messages, indicating that it is deploying:

```
INFO [org.jboss.as.server.deployment] (MSC service thread 1-1)
Starting deployment of "jboss-seam-booking.ear"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-3)
Starting deployment of "jboss-seam-booking.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-6)
Starting deployment of "jboss-seam.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-2)
Starting deployment of "jboss-seam-booking.war"
```

   At this point, you encounter your first deployment error. In the next step, you walk through each issue and learn how to debug and resolve it.

   To learn how to debug and resolve deployment issues, click here: Section 4.3.6, “Debug and Resolve Seam 2.2 Booking Archive Deployment Errors and Exceptions”

   To return to the previous topic, click here: Section 4.3.4, “Migrate the Seam 2.2 Booking Archive to JBoss EAP 6: Step-By-Step Instructions”

Report a bug

4.3.6. Debug and Resolve Seam 2.2 Booking Archive Deployment Errors and Exceptions

In the previous step, Section 4.3.5, “Build and Deploy the JBoss EAP 5.1 Version of the Seam 2.2 Booking Application”, you built the JBoss EAP 5.1 Seam 2.2 Booking application and deployed it to the

JBoss EAP 6 deployment folder. In this step, you debug and resolve each deployment error you encounter.

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

Procedure 4.10. Debug and resolve deployment errors and exceptions

1. Issue - java.lang.ClassNotFoundException: javax.faces.FacesException

   When you deploy the application, the log contains the following error:

   ```
   ERROR \[org.jboss.msc.service.fail\] (MSC service thread 1-1)
   MSC00001: Failed to start service jboss.deployment.subunit."jboss-
   seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
   org.jboss.msc.service.StartException in service
   jboss.deployment.subunit."jboss-seam-booking.ear"."jboss-seam-
   booking.war".POST_MODULE:
   Failed to process phase POST_MODULE of subdeployment "jboss-seam-
   booking.war" of deployment "jboss-seam-booking.ear"
       (.. additional logs removed ...)
   Caused by: java.lang.ClassNotFoundException:
   javax.faces.FacesException from \[Module "deployment.jboss-seam-
   booking.ear:main" from Service Module Loader\]
       at
   org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java
   :191)
   ```

   What it means:

   The ClassNotFoundException indicates a missing dependency. In this case, it cannot find the class **`javax.faces.FacesException`** and you need to explicitly add the dependency.

   How to resolve it:

   Find the module name for that class in the ***`EAP6_HOME`*`/modules/system/layers/base/`** directory by looking for a path that matches the missing class. In this case, you find 2 modules that match:

   ```
   javax/faces/api/main
   javax/faces/api/1.2
   ```

   Both modules have the same module name: **`javax.faces.api`** but one in the main directory is for JSF 2.0 and the one located in the 1.2 directory is for JSF 1.2. If there was only one module available, you could simply create a **`MANIFEST.MF`** file and added the module dependency. But in this case, you want to use the JSF 1.2 version and not the 2.0 version in main, so you need to specify one and exclude the other. To do this, you create a **`jboss-deployment-structure.xml`** file in the EAR's **`META-INF/`** directory that contains the following data:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

In the **deployment** section, you add the dependency for the **javax.faces.api** for the JSF 1.2 module. You also add the dependency for the JSF 1.2 module in the subdeployment section for the WAR and exclude the module for JSF 2.0.

Redeploy the application by deleting the ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** file and creating a blank **jboss-seam-booking.ear.dodeploy** file in the same directory.

2. Issue - java.lang.ClassNotFoundException: org.apache.commons.logging.Log

   When you deploy the application, the log contains the following error:

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-8)
MSC00001: Failed to start service jboss.deployment.unit."jboss-seam-
booking.ear".INSTALL:
org.jboss.msc.service.StartException in service
jboss.deployment.unit."jboss-seam-booking.ear".INSTALL:
Failed to process phase INSTALL of deployment "jboss-seam-
booking.ear"
    (.. additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log from [Module "deployment.jboss-seam-
booking.ear.jboss-seam-booking.war:main" from Service Module Loader]
```

   What it means:

   The **ClassNotFoundException** indicates a missing dependency. In this case, it cannot find the class **org.apache.commons.logging.Log** and you need to explicitly add the dependency.

   How to resolve it:

   Find the module name for that class in the ***EAP6_HOME*/modules/system/layers/base/** directory by looking for a path that matches the missing class. In this case, you find one module that matches the path **org/apache/commons/logging/**. The module name is “org.apache.commons.logging”.

Modify the **jboss-deployment-structure.xml** file to add the module dependency to the deployment section of the file.

```
<module name="org.apache.commons.logging" export="true"/>
```

The **jboss-deployment-structure.xml** should now look like this:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Redeploy the application by deleting the ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** file and creating a blank **jboss-seam-booking.ear.dodeploy** file in the same directory.

3. Issue - java.lang.ClassNotFoundException: org.dom4j.DocumentException

   When you deploy the application, the log contains the following error:

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-3) Exception sending
context initialized event to listener instance of class
org.jboss.seam.servlet.SeamListener: java.lang.NoClassDefFoundError:
org/dom4j/DocumentException
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.dom4j.DocumentException from [Module "deployment.jboss-seam-
booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

   What it means:

   The **ClassNotFoundException** indicates a missing dependency. In this case, it cannot find the class **org.dom4j.DocumentException**.

   How to resolve it:

   Find the module name in the ***EAP6_HOME*/modules/system/layers/base/** directory by looking for the **org/dom4j/DocumentException**. The module name is “org.dom4j”. Modify the **jboss-deployment-structure.xml** file to add the module dependency to the deployment section of the file.

```
<module name="org.dom4j" export="true"/>
```

The **jboss-deployment-structure.xml** file should now look like this:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
            <module name="org.dom4j" export="true"/>
          </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Redeploy the application by deleting the ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** file and creating a blank **jboss-seam-booking.ear.dodeploy** file in the same directory.

4. Issue - java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue

   When you deploy the application, the log contains the following error:

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-6) Exception sending
context initialized event to listener instance of class
org.jboss.seam.servlet.SeamListener: java.lang.RuntimeException:
Could not create Component:
org.jboss.seam.international.statusMessages
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.hibernate.validator.InvalidValue from [Module "deployment.jboss-
seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

   What it means:

   The **ClassNotFoundException** indicates a missing dependency. In this case, it cannot find the class **org.hibernate.validator.InvalidValue**.

   How to resolve it:

   There is a module “org.hibernate.validator”, but the JAR does not contain the **org.hibernate.validator.InvalidValue** class, so adding the module dependency does not resolve this issue. In this case, the JAR containing the class was part of the JBoss EAP 5.1

deployment. Look for the JAR that contains the missing class in the ***EAP5_HOME/seam/lib/*** directory. To do this, open a console and type the following:

```
$ cd EAP5_HOME/seam/lib
$ grep 'org.hibernate.validator.InvalidValue' `find . -name '*.jar'`
```

The result shows:

```
$ Binary file ./hibernate-validator.jar matches
$ Binary file ./test/hibernate-all.jar matches
```

In this case, copy the **hibernate-validator.jar** to the **jboss-seam-booking.ear/lib/** directory:

```
$ cp EAP5_HOME/seam/lib/hibernate-validator.jar jboss-seam-
booking.ear/lib
```

Redeploy the application by deleting the ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** file and creating a blank **jboss-seam-booking.ear.dodeploy** file in the same directory.

5. Issue - java.lang.InstantiationException: org.jboss.seam.jsf.SeamApplicationFactory

   When you deploy the application, the log contains the following error:

```
INFO  [javax.enterprise.resource.webcontainer.jsf.config] (MSC
service thread 1-7) Unsanitized stacktrace from failed start...:
com.sun.faces.config.ConfigurationException: Factory
'javax.faces.application.ApplicationFactory' was not configured
properly.
  at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:296) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
  (... additional logs removed ...)
Caused by: javax.faces.FacesException:
org.jboss.seam.jsf.SeamApplicationFactory
  at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.jav
a:606) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
  (... additional logs removed ...)
  at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:294) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
  ... 11 more
Caused by: java.lang.InstantiationException:
org.jboss.seam.jsf.SeamApplicationFactory
  at java.lang.Class.newInstance0(Class.java:340) [:1.6.0_25]
  at java.lang.Class.newInstance(Class.java:308) [:1.6.0_25]
  at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.jav
a:604) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
  ... 16 more
```

What it means:

The **com.sun.faces.config.ConfigurationException** and **java.lang.InstantiationException** indicate a dependency issue. In this case, the cause is not as obvious.

How to resolve it:

You need to find the module that contains the **com.sun.faces** classes. While there is no **com.sun.faces** module, there are two **com.sun.jsf-impl** modules. A quick check of the **jsf-impl-1.2_13.jar** in the 1.2 directory shows it contains the **com.sun.faces** classes. As you did with the **javax.faces.FacesException ClassNotFoundException**, you want to use the JSF 1.2 version and not the JSF 2.0 version in main, so you need to specify one and exclude the other. You need to modify the **jboss-deployment-structure.xml** to add the module dependency to the deployment section of the file. You also need to add it to the WAR subdeployment and exclude the JSF 2.0 module. The file should now look like this:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
              <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
        <module name="org.dom4j" export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
              <module name="com.sun.jsf-impl" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Redeploy the application by deleting the ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** file and creating a blank **jboss-seam-booking.ear.dodeploy** file in the same directory.

6. Issue - java.lang.ClassNotFoundException: org.apache.commons.collections.ArrayStack

   When you deploy the application, the log contains the following error:

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-1) Exception sending
context initialized event to listener instance of class
com.sun.faces.config.ConfigureListener: java.lang.RuntimeException:
com.sun.faces.config.ConfigurationException: CONFIGURATION FAILED!
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
    (... additional logs removed ...)
```

```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
```

What it means:

The **`ClassNotFoundException`** indicates a missing dependency. In this case, it cannot find the class **`org.apache.commons.collections.ArrayStack`**.

How to resolve it:

Find the module name in the ***`EAP6_HOME`*`/modules/system/layers/base/`** directory by looking for the **`org/apache/commons/collections`** path. The module name is "org.apache.commons.collections". Modify the **`jboss-deployment-structure.xml`** to add the module dependency to the deployment section of the file.

```
<module name="org.apache.commons.collections" export="true"/>
```

The **`jboss-deployment-structure.xml`** file should now look like this:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
              <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
        <module name="org.dom4j" export="true"/>
        <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
              <module name="com.sun.jsf-impl" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Redeploy the application by deleting the ***`EAP6_HOME`*`/standalone/deployments/jboss-seam-booking.ear.failed`** file and creating a blank **`jboss-seam-booking.ear.dodeploy`** file in the same directory.

7. Issue - Services with missing/unavailable dependencies

   When you deploy the application, the log contains the following error:

```
ERROR [org.jboss.as.deployment] (DeploymentScanner-threads - 2)
{"Composite operation failed and was rolled back. Steps that
failed:" => {"Operation step-2" => {"Services with
```

```
missing/unavailable dependencies" =>
["jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.AuthenticatorAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]","jboss.deployment.subunit.\"jboss-
seam-booking.ear\".\"jboss-seam-
booking.jar\".component.HotelSearchingAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".HotelSearchingAction.\"env/org.jboss.seam.example.book
ing.HotelSearchingAction/em\" ]","
  (... additional logs removed ...)
"jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.BookingListAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".BookingListAction.\"env/org.jboss.seam.example.booking
.BookingListAction/em\" ]","jboss.persistenceunit.\"jboss-seam-
booking.ear/jboss-seam-booking.jar#bookingDatabase\" missing [
jboss.naming.context.java.bookingDatasource ]"]}}}
```

What it means:

When you get a “Services with missing/unavailable dependencies” error, look at the text within the brackets after “missing”. In this case you see:

```
missing [ jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-
seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]
```

The “/em” indicates an Entity Manager and datasource issue.

How to resolve it:

In JBoss EAP 6, datasource configuration has changed and needs to be defined in the ***EAP6_HOME*/standalone/configuration/standalone.xml** file. Because JBoss EAP 6 ships with an example database that is already defined in the **standalone.xml** file, modify the **persistence.xml** file to use that example database in this application. Looking in the **standalone.xml** file, you can see that the **jndi-name** for the example database is **java:jboss/datasources/ExampleDS**. Modify the **jboss-seam-booking.jar/META-INF/persistence.xml** file to comment the existing **jta-data-source** element and replace it as follows:

```
<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

Redeploy the application by deleting the ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** file and creating a blank **jboss-seam-booking.ear.dodeploy** file in the same directory.

8. At this point, the application deploys without errors, but when you access the URL http://localhost:8080/seam-booking/ in a browser and attempt "Account Login", you get a runtime error “The page isn't redirecting properly”. In the next step, you learn how to debug and resolve runtime errors.

To learn how to debug and resolve runtime issues, click here: Section 4.3.7, “Debug and Resolve Seam 2.2 Booking Archive Runtime Errors and Exceptions”

To return to the previous topic, click here: Section 4.3.4, “Migrate the Seam 2.2 Booking Archive to JBoss EAP 6: Step-By-Step Instructions”

Report a bug

4.3.7. Debug and Resolve Seam 2.2 Booking Archive Runtime Errors and Exceptions

In the previous step, Section 4.3.6, “Debug and Resolve Seam 2.2 Booking Archive Deployment Errors and Exceptions”, you learned how to debug deployment errors. In this step, you debug and resolve each runtime error you encounter.

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

Procedure 4.11. Debug and resolve runtime errors and exceptions

At this point, when you deploy the application you do not see any errors in the log. However, when you access the application URL, errors appear in the log.

1. Issue - javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not found in context ''

   When you access the URL http://localhost:8080/seam-booking/ in a browser, you get "The page isn't redirecting properly" and the log contains the following error:

   ```
   SEVERE [org.jboss.seam.jsf.SeamPhaseListener] (http--127.0.0.1-8080-
   1) swallowing exception: java.lang.IllegalStateException: Could not
   start transaction
     at
   org.jboss.seam.jsf.SeamPhaseListener.begin(SeamPhaseListener.java:59
   8) [jboss-seam.jar:]
     (... log messages removed ...)
   Caused by: org.jboss.seam.InstantiationException: Could not
   instantiate Seam component:
   org.jboss.seam.transaction.synchronizations
     at org.jboss.seam.Component.newInstance(Component.java:2170)
   [jboss-seam.jar:]
     (... log messages removed ...)
   Caused by: javax.naming.NameNotFoundException: Name 'jboss-seam-
   booking' not found in context ''
     at
   org.jboss.as.naming.util.NamingUtils.nameNotFoundException(NamingUti
   ls.java:109)
     (... log messages removed ...)
   ```

   What it means:

A **NameNotFoundException** indicates a JNDI naming issue. JNDI naming rules have changed in JBoss EAP 6, so you need to modify the lookup names to follow the new rules.

How to resolve it:

To debug this, look earlier in the server log trace to what JNDI binding were used. Looking at the server log you see this:

```
15:01:16,138 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUn
itProcessor] (MSC service thread 1-1) JNDI bindings for session bean
named RegisterAction in deployment unit subdeployment "jboss-seam-
booking.jar" of deployment "jboss-seam-booking.ear" are as follows:
  java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
  java:app/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
  java:module/RegisterAction!org.jboss.seam.example.booking.Register
  java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction
  java:app/jboss-seam-booking.jar/RegisterAction
  java:module/RegisterAction
  [JNDI bindings continue ...]
```

There are a total of eight INFO JNDI bindings listed in the log, one for each session bean: RegisterAction, BookingListAction, HotelBookingAction, AuthenticatorAction, ChangePasswordAction, HotelSearchingAction, EjbSynchronizations, and TimerServiceDispatcher. You need to modify the WAR's **lib/components.xml** file to use the new JNDI bindings. In the log, note the EJB JNDI bindings all start with "java:app/jboss-seam-booking.jar" Replace the **core:init** element as follows:

```
<!--     <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{ejbName}"
debug="true" distributable="false"/>
```

Next, you need to add the EjbSynchronizations and TimerServiceDispatcher JNDI bindings. Add the following component elements to the file:

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
<component class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

The components.xml file should now look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
  xmlns:core="http://jboss.com/products/seam/core"
  xmlns:security="http://jboss.com/products/seam/security"
  xmlns:transaction="http://jboss.com/products/seam/transaction"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/core
```

```
http://jboss.com/products/seam/core-2.2.xsd
     http://jboss.com/products/seam/transaction
http://jboss.com/products/seam/transaction-2.2.xsd
     http://jboss.com/products/seam/security
http://jboss.com/products/seam/security-2.2.xsd
     http://jboss.com/products/seam/components
http://jboss.com/products/seam/components-2.2.xsd">

    <!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
    <core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>
    <core:manager conversation-timeout="120000"
                  concurrent-request-timeout="500"
                  conversation-id-parameter="cid"/>
    <transaction:ejb-transaction/>
    <security:identity authenticate-method="#
{authenticator.authenticate}"/>
    <component
class="org.jboss.seam.transaction.EjbSynchronizations"
            jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
    <component class="org.jboss.seam.async.TimerServiceDispatcher"
            jndi-name="java:app/jboss-
seam/TimerServiceDispatcher"/>
</components>
```

Redeploy the application by deleting the **`standalone/deployments/jboss-seam-booking.ear.failed`** file and creating a blank **`jboss-seam-booking.ear.dodeploy`** file in the same directory.

2. Issue - The application deploys and runs without error. When you access the URL http://localhost:8080/seam-booking/ in a browser and attempt to login, it fails with the message "Login failed. Transaction failed." You should see an exception trace in the server log:

```
13:36:04,631 WARN  [org.jboss.modules] (http-/127.0.0.1:8080-1)
Failed to define class
org.jboss.seam.persistence.HibernateSessionProxy in Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
....
Caused by: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
...
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/engine/SessionImplementor
 at java.lang.ClassLoader.defineClass1(Native Method)
[rt.jar:1.7.0_45]
...
Caused by: java.lang.ClassNotFoundException:
org.hibernate.engine.SessionImplementor from [Module
```

```
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader]
...
```

What it means:

The ClassNotFoundException indicates a missing Hibernate library. In this case it's the **`hibernate-core.jar`**.

How to resolve it:

Copy the **`hibernate-core.jar`** JAR from the **`EAP5_HOME/seam/lib/`** directory to the **`jboss-seam-booking.ear/lib`** directory.

Redeploy the application by deleting the **`standalone/deployments/jboss-seam-booking.ear.failed`** file and creating a blank **`jboss-seam-booking.ear.dodeploy`** file in the same directory.

3. Issue - The application deploys and runs without error. When you access the URL http://localhost:8080/seam-booking/ in a browser, you are able to login successfully. However, when you attempt to book a hotel, you will see an exception trace.

   To debug this, you must first remove the **`jboss-seam-booking.ear/jboss-seam-booking.war/WEB-INF/lib/jboss-seam-debug.jar`** as it masks the true error. At this point, you should see the following error:

   ```
   java.lang.NoClassDefFoundError:
   org/hibernate/annotations/common/reflection/ReflectionManager
   ```

   What it means:

   The NoClassDefFoundError indicates a missing Hibernate library.

   How to resolve it:

   Copy the **`hibernate-annotations.jar`** and **`hibernate-commons-annotations.jar`** JARs from the **`EAP5_HOME/seam/lib/`** directory to the **`jboss-seam-booking.ear/lib`** directory.

   Redeploy the application by deleting the **`standalone/deployments/jboss-seam-booking.ear.failed`** file and creating a blank **`jboss-seam-booking.ear.dodeploy`** file in the same directory.

4. Runtime and application errors should be resolved

   At this point, the application deploys and runs without error.

   To return to the previous topic, click here: Section 4.3.4, “Migrate the Seam 2.2 Booking Archive to JBoss EAP 6: Step-By-Step Instructions”

Report a bug

4.3.8. Review a Summary of the Changes Made When Migrating the Seam 2.2 Booking Application

Although it would be much more efficient to determine dependencies in advance and add the implicit dependencies in one step, this exercise shows how problems appear in the log and provides some

information on how to debug and resolve them. The following is a summary of changes made to the application when migrating it to JBoss EAP 6.

IMPORTANT

Applications that use Hibernate directly with Seam 2.2 may use a version of Hibernate 3 packaged inside the application. Hibernate 4, which is provided through the org.hibernate module of JBoss EAP 6, is not supported by Seam 2.2. This example is intended to help you get your application running on JBoss EAP 6 as a first step. Please be aware that packaging Hibernate 3 with a Seam 2.2 application is not a supported configuration.

1. You created a **`jboss-deployment-structure.xml`** file in the EAR's **`META-INF/`** directory. You added **`<dependencies>`** and **`<exclusions>`** to resolve **`ClassNotFoundExceptions`**. This file contains the following data:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
     <dependencies>
       <module name="javax.faces.api" slot="1.2" export="true"/>
       <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
       <module name="org.apache.commons.logging" export="true"/>
           <module name="org.dom4j" export="true"/>
       <module name="org.apache.commons.collections"
export="true"/>
     </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
   <exclusions>
       <module name="javax.faces.api" slot="main"/>
       <module name="com.sun.jsf-impl" slot="main"/>
     </exclusions>
     <dependencies>
       <module name="javax.faces.api" slot="1.2"/>
       <module name="com.sun.jsf-impl" slot="1.2"/>
     </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

2. You copied the following JARs from the ***`EAP5_HOME`*`/jboss-eap-5.1/seam/lib/`** directory to the **`jboss-seam-booking.ear/lib/`** directory to resolve **`ClassNotFoundExceptions`**:
   - hibernate-core.jar
   - hibernate-validator.jar
3. You modified the **`jboss-seam-booking.jar/META-INF/persistence.xml`** file as follows.
   1. You changed the **`jta-data-source`** element to use the Example database that ships with JBoss EAP 6:

```
<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -
->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
```

```
source>
```

2. You commented out the hibernate.cache.provider_class property:

```
<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

4. You modified the WAR's **lib/components.xml** file to use the new JNDI bindings

1. You replaced the **core:init** existing element as follows:

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>
```

2. You added component elements for the "EjbSynchronizations" and "TimerServiceDispatcher" JNDI bindings

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
 <component class="org.jboss.seam.async.TimerServiceDispatcher"
jndi-name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

Report a bug

APPENDIX A. REVISION HISTORY

Revision 2.0-21 **Tue Mar 4 2014** **Russell Dickenson**

Red Hat JBoss Enterprise Application Platform 6.2.0 GA