Red Hat Application Server

JOnAS Tutorial

Red Hat Application Server: JOnAS Tutorial
Copyright © 1999-2005 by ObjectWeb Consortium

Table of Contents

![redhat logo]

Chapter 1.
Introduction

This tutorial explains how to configure and run the Red Hat Application Server server. In addition, it explains how to run the examples provided in the Red Hat Application Server distribution. It is assumed that a JDK is already installed. For installation details, refer to the Red Hat Application Server *Installation Guide*.

1.1. About this Guide

This introduction gives an overview of Red Hat Application Server. For those who want to immediately start using Red Hat Application Server without going through the overview, you can go to Chapter 2 *Quick Start*.

Chapter 3 *Configuring Your Environment* helps you set up all the software needed for a complete J2EE environment, including a *Web server*, *Servlet*, and *EJB (Enterprise JavaBean)* servers.

Chapter 4 *Getting Started With JOnAS* is a quick reference to JOnAS that guides you through the most common tasks you will have to perform.

The remaining chapters are specific to EJB application development. Each chapter treats a different programming concept and illustrates it with a specific example. The **Alarm** application is a "putting it all together" example that gives an overview of a complete application using JOnAS.

1.2. Enterprise JavaBeans

The Sun Enterprise JavaBeans specification (http://java.sun.com/products/ejb/docs.html) defines an architecture and interfaces for developing and deploying distributed Java server applications based on a multi-tier architecture.

The intent of this specification is to facilitate and normalize the development, deployment, and assembly of application components (called enterprise beans); such components will be deployable on EJB platforms. The resulting applications are typically transactional, database-oriented, multi-user, secured, scalable, and portable. More precisely, this EJB specification addresses the following areas:

- The runtime environment, called the EJB server, which provides the execution environment together with the transactional service, the distribution mechanisms, the persistence management, and the security.
- A programmer and user guide explaining how an enterprise bean should be developed, deployed, and used.

Not only will an enterprise bean be independent of the platform and operating system (since it is written in Java), but also of the EJB platform.

1.2.1. What Does What?

The Java2 Enterprise Edition (J2EE) platform is an *n*-tier platform. Each tier has a specific role:

Clients. Clients are the end-users of the system and can access the J2EE server by several means. One of the most common is to use a Web browser such as **Mozilla** to connect to a Web server where Java Servlets or Java Server Pages (JSP) access the business logic of the J2EE server (see Figure 1-1). Another solution is to use a specific Java program that directly communicates with the J2EE server (see Figure 1-2).

Web server. The Web server is responsible for accepting client requests and sending HTML replies back to the client. HTML pages can be static files that reside on the Web server filesystem or dynamically built with Servlets or JSPs from data generated by the beans.

Servlet/JSP server. The Java Server Page (JSP) server or Servlet container can be integrated into the Web server or be a separate entity communicating with the Web server. JSP or servlets run within a Java Virtual Machine that can be the same as the one used by the J2EE server.

J2EE application server. The J2EE application server is the place where the beans are executed. JOnAS is fully implemented in Java and all the beans loaded in an instance of JOnAS run within the same Java Virtual Machine. It is possible to run several different JOnAS J2EE application servers on the same or separate machines.

Database server. The database server is used to store and retrieve data. It is accessed through the standard *JDBC (Java DataBase Connectivity)* API by the beans.
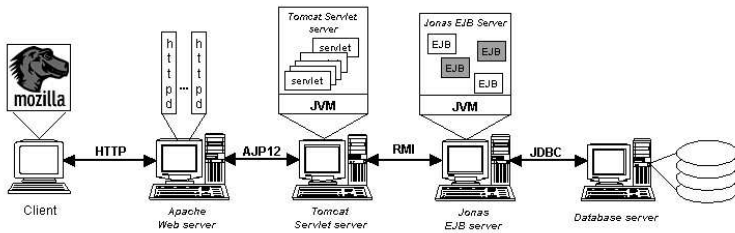

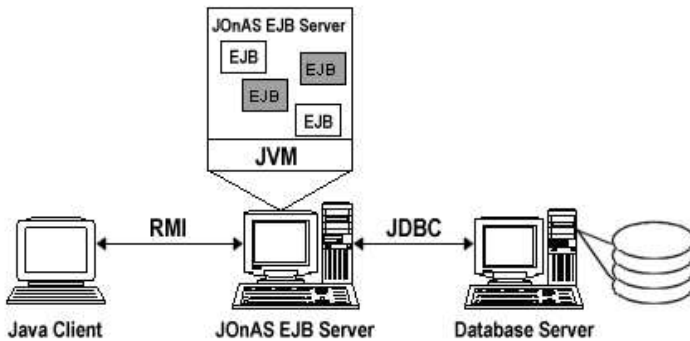**Figure 1-1. J2EE *n*-tier architecture overview**

Figure 1-2. Accessing the Red Hat Application Server server business logic from a Java client

The distributed environment in the EJB world uses RMI, but JOnAS supports several distributed processing environments:

• RMI using the Sun proprietary protocol JRMP

- Jeremie: the RMI personality of an Object Request Broker called Jonathan, which also provides a *CORBA (Common Object Request Broker Architecture)* personality. Jeremie allows JOnAS to benefit from the optimization of local RMI calls.
- RMI/IIOP: JOnAS now provides support for both *RMI/IIOP (Java Remote Method Invocation over Internet Inter-Orb Protocol)* and RMI/JRMP by integrating the CAROL communication framework.
- *CMI (Cluster Method Invocation)*: A new *ORB (Object Request Broker)* used by JOnAS to provide clustering for load balancing and high availability.

1.2.2. JOnAS Features

JOnAS implements a set of J2EE specifications. The following list shows the specifications that are implemented by JOnAS and gives a brief description of the services provided by JOnAS.

1.2.2.1. Specifications

JOnAS provides full support of the following specifications:

- *EJB (Enterprise JavaBean)* (http://java.sun.com/products/ejb): Enterprise JavaBeans containers are provided by a set of Java classes and a tool to generate interposition classes.
- *JTA (Java Transaction API)* (http://java.sun.com/products/jta): a Transaction Manager that provides Java Transaction API support and distributed transaction coordination.
- *JDBC (Java DataBase Connectivity)* (http://java.sun.com/products/jdbc): a Database Manager that provides Java DataBase Connectivity support.
- *JMS (Java Messaging Service)* (http://java.sun.com/products/jms): Java Messaging Service is provided by JORAM, a technology from ScalAgent (http://www.scalagent.com).
- *JMX (Java Management Extension)* (http://java.sun.com/products/JavaManagement): JOnAS contains the Java Management Extension technology. Management and monitoring is available through a Web interface.
- *J2EE CA (J2EE Connector Architecture)* (http://java.sun.com/j2ee/connector): JOnAS supports the J2EE Connector Architecture that defines a set of mechanism that enable the integration of Enterprise Information Systems (EIS).
- *JNDI (Java Naming and Directory Interface)* (http://java.sun.com/products/jndi): Provides naming and directory functionality.
- Security management.

1.2.2.2. JOnAS Services

JOnAS offers several services that can be turned on or off as required:

- EJB Container Service: A set of Java classes implementing the EJB specification.
- Web Container Service: A servlet/JSP engine in the *JVM (Java Virtual Machine)* of the Red Hat Application Server server and the loading of web applications ("WAR files") within this engine. Currently this service can be configured to use *Tomcat* or *Jetty*, although only Tomcat is supplied with Red Hat Application Server.
- EAR Service: A service used for deploying complete J2EE applications; that is, applications packaged in EAR files, which themselves contain ejb-jar files and/or WAR files.

- JDBC Service: JDBC 2.0 support including *XA (Distributed transaction mode for JDBC 2.0)* resources and connection pooling.
- Security Service: This service implements the authorization mechanisms for accessing EJB components, as specified in the EJB specification. EJB security is based on the notion of roles.
- Transaction Service: This is a Java Transaction Monitor called JOnAS JTM, which ensures the coordination of distributed transactions using XA. It handles two-phase commit protocol against any number of Resource Managers (XA Resources).
- Messaging Service: This service is in charge of launching (or establishing connection to) an integrated JMS server. JOnAS makes use of a third-party JMS implementation. Currently the JORAM open-source JMS server is integrated and delivered with JOnAS.
- Management Service: Integration of a JMX server.
- J2EE CA Resource Service: Allows application component access to an external EIS.
- Mail Service: Allows application components to read or send e-mail using JavaMail.
- Database Service: This service is responsible for handling Datasource objects. A Datasource is a standard JDBC administrative object for handling connections to a database.
- Communication and Naming Service: This service provides the JNDI API to application components and to other services to bind and lookup remote objects (for example, EJB Homes) and resource references (JDBC Datasource, Mail, and JMS connection factories, etc.)

Configuring and starting the various services is described in Section 4.1.2 *JOnAS Configuration Files*.

1.2.3. Software Requirements

Here is a list of the main software components needed to run Red Hat Application Server:

- Red Hat Enterprise Linux 4.
- A Java2 Software Development Kit (SDK) version 1.4. These are available on the RHEL4 Extras Channel.

Here is a list of software components that may also be needed to run JOnAS in some configurations:

- A Web server may be used in front of the Red Hat Application Server application server. The most popular Open Source web server software used is Apache.
- A Database server may be needed. Any database with a JDBC driver can be used with Red Hat Application Server. Red Hat Application Server is configured to use PostgreSQL (which is available on the RHEL4 Base Channel), although any database with a JDBC driver can be used.

Chapter 2.
Quick Start

This chapter will help you set up Red Hat Application Server and run your first EJB application. If you do not understand all the steps, you should read the next two chapters to familiarize yourself with the details of setting up Red Hat Application Server.

2.1. Setting Up the Red Hat Application Server Environment

We assume that your JDK is already installed and configured.

Once you have installed your Red Hat Application Server distribution, you have to set up the `JONAS_ROOT` environment variable prior to using JOnAS or any of its tools. You will also have to update your `PATH variable` as well.

Open a new terminal and proceed as follows:

```
bash> export JONAS_ROOT=/usr/share/jonas
bash> export PATH=${PATH}:${JONAS_ROOT}/bin/unix
```

2.2. Running Your First EJB Application

Several example programs are included with the JOnAS distribution. They are located in the `$JONAS_ROOT/examples/src` directory. The one we will use as your first EJB application is the Session Bean (SB) example that resides in `$JONAS_ROOT/examples/src/sb`.

The SB example involves a Java client that accesses a *Stateful Session Bean* and twice invokes the `buy` method of the bean transaction. The client communicates with JOnAS using RMI and the configuration files should be already set so that the RMI registry will be automatically started and embedded in the same JVM as JOnAS. The figure below gives an overview of this application.
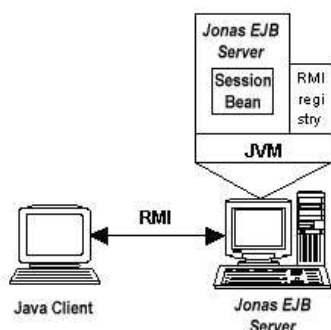
Figure 2-1. EB example overview

2.2.1. Building the Examples

The simplest way to compile all examples is to start in the `$JONAS_ROOT/examples/src` directory and enter the following command line as user `jonas`:

```
ant -find build.xml install
```

This command compiles all examples in the `$JONAS_ROOT/examples/src` directory.

Note

All examples assume that the current directory is `$JONAS_ROOT/examples/src/` followed by the directory that contains the example. In other words, the commands entered for the SB example assume the current folder is `$JONAS_ROOT/examples/src/sb`, whereas the commands entered for the EB example assume the current folder is `$JONAS_ROOT/examples/src/eb`.

To login as user `jonas`, first login as `root`, then `su` to user `jonas`, specifying `/bin/bash` as the shell:

```
su -s /bin/bash jonas
```

2.2.2. Quick Start to the SB Example

Note

This example is described in more detail in Section 5.3 *Running the SB Example*.

To run this example, you will have to first start the Red Hat Application Server server and then run the Java client. Finally, at the end of the execution you should stop the Red Hat Application Server server.

Note

The following example assumes that the current directory is `$JONAS_ROOT/examples/src/sb`.

Here is how to proceed:

1. As `root`, start the Red Hat Application Server server:
   /sbin/service jonas start

2. Deploy the JAR file:
   - To deploy the JAR file from the command line:
     jonas admin -a sb.jar
   - To deploy the JAR file using the web interface:

     a. Log into the JOnAS Administrator.
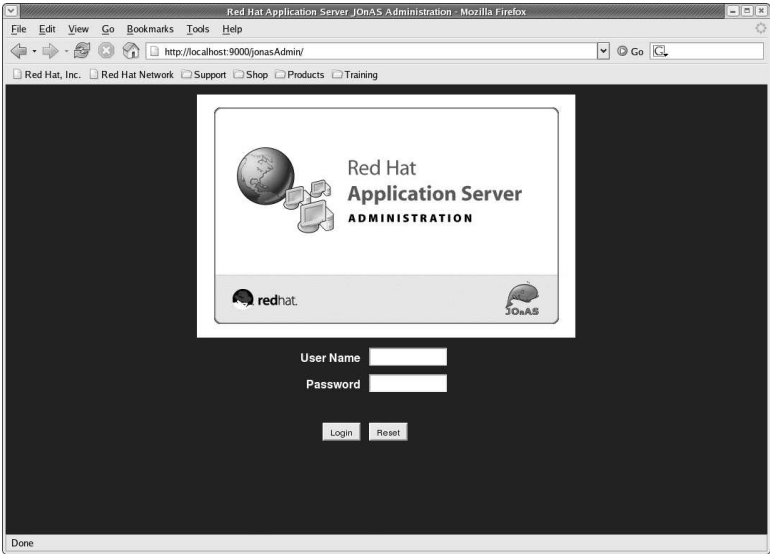
Figure 2-2. JOnAS Administration Login

Log in with User Name jonas and Password jonas.
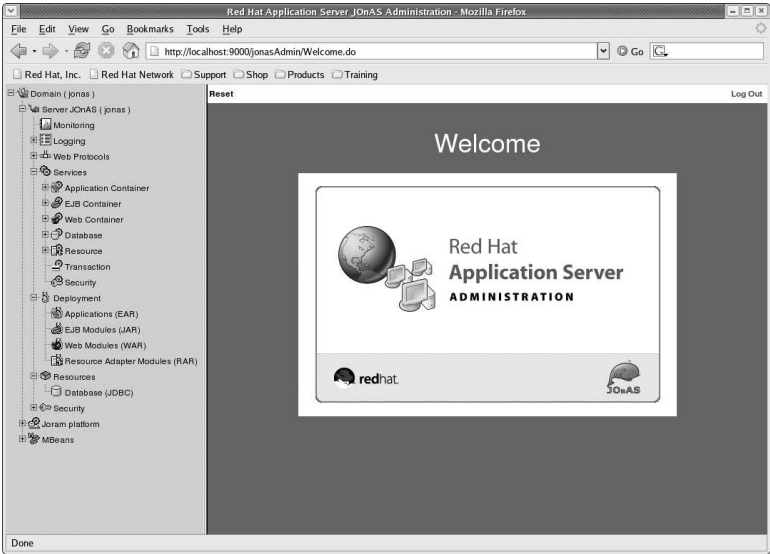
Figure 2-3. JOnAS Administration Web Interface

b. Click **Deployments > EJB Modules (JAR)**.
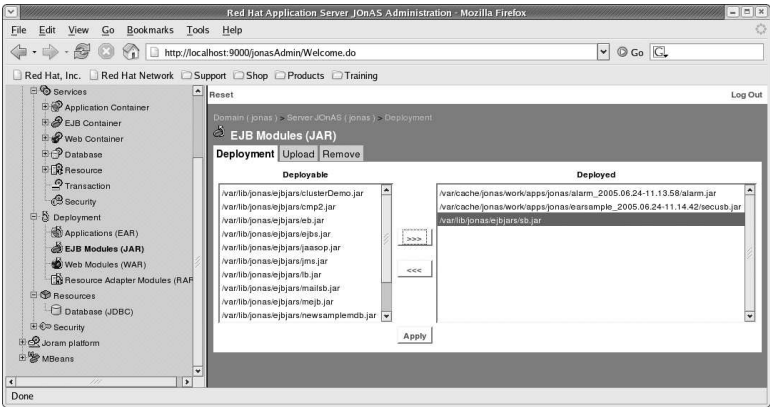
Figure 2-4. Deploy the sb.jar Example

c. On the `sb.jar` in the **Deployable** list box on the left side, Click **Deploy**, then click **Apply**.

d. A **Confirm** dialog appears; click **Confirm**.
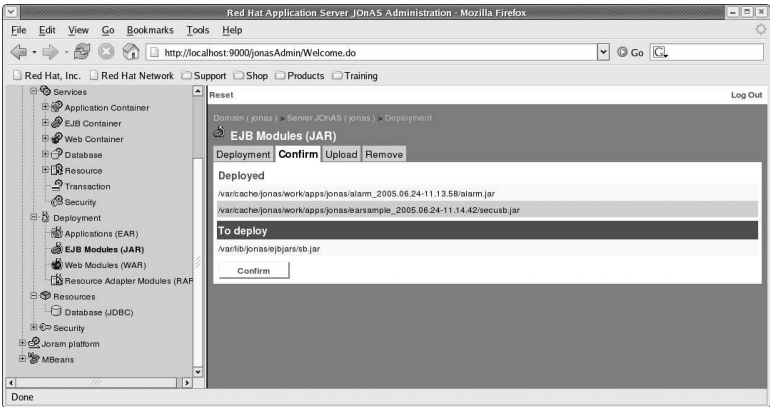
Figure 2-5. Confirm the sb.jar Deployment

e. A **Result** dialog appears, and the newly deployed `jar` is in the list.
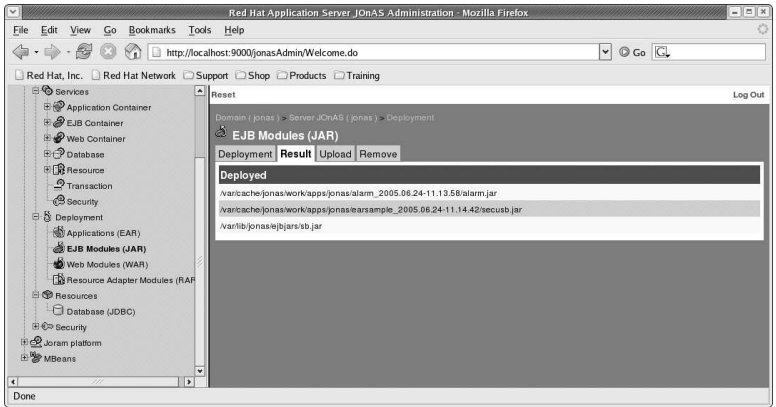
Figure 2-6. Results of the sb.jar Deployment

    f. If you click **Deployments > EJB Modules (JAR)** again, `sb.jar` will appear in the right list box, which lists the deployed `jar` files.

3. Start the Java Client:
   `jclient sb.ClientOp`

   A successful run should output:
   ```
   Create a bean
   Start a first transaction
   First request on the new bean
   Second request on the bean
   Commit the transaction
   Start a second transaction
   Rollback the transaction
   Request outside any transaction
   ClientOp OK. Exiting.
   ```

   Congratulations! You have succeeded running your first EJB application with JOnAS!

4. As `root`, stop the Red Hat Application Server server with the following command:
   `/sbin/service jonas stop`

Configuring Your Environment

In this chapter, we will put all the pieces together that are required to configure a complete J2EE platform. Note that you can find a more detailed description in the documentation provided with each software component.

3.1. Setting Up Your Java Environment

To set up your Java environment, you must set the value for JAVA_HOME and update the PATH environment variable. JAVA_HOME must point to the SDK directory and PATH must point to the bin subdirectory of the SDK.

- Assuming that you installed the SDK in the /usr/lib/jvm/jre-1.4.2-bea/bin/java directory, you should enter the following:
  bash> **export JAVA_HOME=/usr/lib/jvm/jre-1.4.2-bea/**
- You then have to update your PATH variable by appending /usr/local/jdk1.4.2/bin. If JAVA_HOME has been properly set, you can proceed as follows:
  bash> **export PATH=${PATH}:${JAVA_HOME}/bin**

Environment variables must be correctly set at all times. It is common practice to insert these commands in your shell start file (~/.bashrc).

3.1.1. Configuring Ant

You have to set the ANT_HOME environment variable and update your PATH variable.

Assuming that you installed Ant in the /usr/share/ant directory, you should enter the following command:

bash> **export ANT_HOME=/usr/share/ant**

Note

Make sure that the ant executable is available through the PATH. (Usually the ant executable is installed in /usr/bin.)

Warning

If you want Ant to automatically set the CLASSPATH variable, you have to correctly set your *JAVA_HOME* environment variable as described in Section 3.1 *Setting Up Your Java Environment*.

3.1.2. Configuring Your JOnAS Environment

You have to set up the JONAS_ROOT environment variable prior to using JOnAS or any of its tools. You also have to update your PATH.

Assuming that you installed JOnAS in the /usr/share/jonas directory, enter the following commands:

```
bash> export JONAS_ROOT=/usr/share/jonas
bash> export PATH=${PATH}:${JONAS_ROOT}/bin/unix
```

Chapter 4.
Getting Started With JOnAS

This chapter will help you to understand the various JOnAS components as well as how to start and manage a JOnAS server.

4.1. Overview of the Tools

JOnAS comes with a set of tools that are briefly described below.

- `/sbin/service jonas start` starts the JOnAS server.
- `newbean` generates skeleton files when developing a new bean.
- `registry` starts the JNDI according to the setting in the `carol.properties` file (usually the setting is RMI or JEREMIE). This command should be used only if you configured JOnAS to use a remote registry.
- `GenIC` generates container classes.
- `JmsServer` starts the JMS server.
- `jonas admin` a command-line administration console for JOnAS.
- `RAConfig` Resource Adapter deployment tool.

All these commands and their respective options are fully detailed in the *Red Hat Application Server User Guide*.

4.1.1. Starting and Stopping JOnAS

If JOnAS has been properly installed and your environment variables are correctly defined as described in Section 3.1.2 *Configuring Your JOnAS Environment*, the JOnAS service is started as follows:

/sbin/service jonas start

A name is assigned to each JOnAS server in the `jonas.properties` file (see Section 4.1.2 *JOnAS Configuration Files*). The default name for a JOnAS server is `jonas`. To stop a JOnAS server, you can use the `jonas` tool. The following command stops the JOnAS server named `jonas` (default):

/sbin/service jonas stop

To stop a JOnAS server with a user defined name such as `myEJBserver`, use:

/sbin/service jonas stop −n myEJBserver

Note

If the registry was launched with the JOnAS server, halting the server also terminates the registry. However, if the registry was launched separately, it is up to you to stop it if needed.

4.1.2. JOnAS Configuration Files

This section is a guide for the JOnAS configuration files. If you need more in-depth information, check the *JOnAS Configuration* chapter of the *Red Hat Application Server User Guide*.

The JOnAS distribution contains configuration files that reside in the `$JONAS_ROOT/conf` directory:

• `jonas.properties` is used to configure the JOnAS server and the different services that it may launch.
• `carol.properties` is used to configure access to JNDI (RMI or JEREMIE).
• Additional properties file used to configure database access.

The JOnAS distribution contains a number of configuration files in the `$JONAS_ROOT/conf` directory. You can edit these files to change the default configuration, but we recommend that you use a different location for the configuration files needed by a specific application running on JOnAS. This is achieved by using an additional environment variable called `JONAS_BASE`.

Additional configuration files may be required for specific features. For those advanced features, refer to the *JOnAS Configuration* chapter of the *Red Hat Application Server User Guide*.

4.1.2.1. jonas.properties

JOnAS first checks `$JONAS_BASE/conf/jonas.properties`. If `$JONAS_BASE` is not defined, it is automatically initialized to `$JONAS_ROOT`. JOnAS then checks in `$JONAS_ROOT/conf/jonas.properties`, then `$HOME/jonas.properties`, and finally in `./jonas.properties`. Files are read in this order each one overriding the values previously defined.

4.1.2.2. carol.properties

Access to JNDI is bound to the `carol.properties` file, which must be accessible from the classpath. This file is supplied with the JOnAS distribution in the `$JONAS_ROOT/conf` directory.

4.1.3. Database Access

Any database with a JDBC driver can be used with JOnAS. Sample properties files are provided for Oracle, InstantDB, Interbase, and PostgreSQL. If you want to setup a new database, you can modify one of those configuration files to suit your database driver requirements. Here is the PostgreSQL example:

```
##################### PostgreSQL DataSource configuration example
#

#####
#  DataSource configuration
#  Replace db_jonas and jonas by appropriate values.
#
datasource.name          jdbc_1
datasource.url           jdbc:postgresql://localhost:5433/db_jonas
datasource.classname     org.postgresql.Driver
datasource.username      jonas
datasource.password      jonas
datasource.mapper rdb.postgres


#####
```

```
#   ConnectionManager configuration
#
#   JDBC connection checking level.
#       0 = no special checking
#       1 = check physical connection is still open before reusing it
#       2 = try every connection before reusing it
jdbc.connchecklevel 1

#   Max age for jdbc connections
#       nb of minutes a connection can be kept in the pool
jdbc.connmaxage  30

#   Test statement
jdbc.connteststmt select 1
```

Note that postgresql_ds is the name that you will have to use in your beans deployment descriptors to reference this datasource.

If you want use the PostgreSQL datasource example above through a JDBC Resource Adapter, you need to create a resource archive (`.rar`) file as follows.

Assuming you have called the configuration file `PostgreSQL1.properties`, and `JONAS_ROOT` is set to the JOnAS installation directory (`/usr/share/jonas` by default), run the following command as root:

```
su – jonas -s /bin/sh -c "cd $JONAS_ROOT/rars/autoload/;                    \
   $JONAS_ROOT/bin/unix/RAConfig -j jdbc_1 -dm -p                           \
   $JONAS_ROOT/conf/PostgreSQL1 $JONAS_ROOT/rars/autoload/JOnAS_jdbcDM      \
   PostgreSQL1_DM"
```

The jndi name to use to access this source would then be `jdbc_1`. You should also ensure that "`resource`" is among the services listed in the `jonas.services` list in your `jonas.properties` file.

Optionally, you can also use the 'dbm' service instead. However, this method is deprecated.

To use the Database Service, you need to modify your `jonas.properties` file:

Assuming you have called the configuration file `PostgreSQL1.properties`, you must define a line such as

```
jonas.service.dbm.datasources PostgreSQL1
```

in your `jonas.properties` file. You should also ensure that "`dbm`" is among the services listed in the `jonas.services` list in your `jonas.properties` file.

4.1.4. Loading Beans Using jonas.properties

You can statically define the beans you want to load at JOnAS startup by updating the `jonas.properties` file. Here are some examples:

- If you want to load the beans defined in a single `ejb-jar.xml` file, and not packaged in an `ejb-jar` file, place this line in your `jonas.properties` file:
  ```
  jonas.service.ejb.descriptors    META-INF/ejb-jar.xml
  ```

  Note that in the above case, the shell `CLASSPATH` used to start the JOnAS server must also allow access to your bean classes.

- To deploy beans defined in several deployment descriptors, the syntax is as follows:
  ```
  jonas.service.ejb.descriptors    Bean1.xml,Bean2.xml
  ```

- If you want to load all the beans of your application, you can give the name of the application jar file, but you must store your xml deployment descriptor in the `META-INF` directory:
  ```
  jonas.service.ejb.descriptors    myApp.jar
  ```

4.1.5. JOnAS Administration

JOnAS can be administered from a Web interface with:

http://*hostname*:9000/jonasAdmin

Login with User Name `jonas` and Password `jonas`.

JOnAS can also be managed using the command-line tool called `jonas`, using username `jonas` and password `jonas`. The administrative option is started by passing `admin` as a command-line argument. For example:

jonas admin

See *JOnAS Command Reference* in the *Red Hat Application Server User Guide*.

4.1.6. Loading Beans Using jonas admin

JOnAS provides a command-line console that allows you to perform many administration tasks such as dynamically loading beans. Here is how to load beans with JOnAS Admin:

- Loading all beans of the application stored in `myApp.jar` into the JOnAS server named *jonas* (default name):
  jonas admin -a myApp.jar
- Load all beans defined in an ejb-jar file into a JOnAS server named *jonas2*:
  jonas admin -n jonas2 -a mybean-ejb.jar
- Same as the previous example but using the interactive console:
  ```
  > jonas admin
  You must first choose a jonas server. (command 'name')
  Type 'help' to get the list of available commands
  jonas admin (jonas) > name jonas2
  jonas admin (jonas2) > addbeans mybean-ejb.jar
  ```

You can find a complete description of JOnAS `Admin` in the *JOnAS Command Reference* section of the *Red Hat Application Server User Guide*.

4.1.7. Unloading Beans

Beans can be unloaded either by stopping JOnAS or using the command line `jonas admin`. Here is how to unload beans with JOnAS `Admin`:

- Unload all beans defined in an ejb-jar file from a JOnAS server named *jonas2*:
  jonas admin -n jonas2 -r mybean-ejb.jar
- Same as the previous example but using the interactive console:
  ```
  > jonas admin
  You must first choose a jonas server. (command 'name')
  Type 'help' to get the list of available commands
  jonas admin (jonas) > name jonas2
  jonas admin (jonas2) > removebeans mybean-ejb.jar
  ```

Session Beans

The JOnAS distribution comes with several example applications. In the following example, we will use a java client that accesses a Stateful Session bean and invokes the `buy` method of the bean transaction twice.

5.1. Finding the Example Application

The Session Bean example application is located in the `$JONAS_ROOT/examples/src/sb` directory. It is composed of the following files:

```
-rw-r--r--  4608 Apr 25  2003 ClientOp.java
-rw-r--r--   512 Apr 25  2003 jonas-sb.xml
-rw-r--r--  7680 Apr 25  2003 OpBean.java
-rw-r--r--  1536 Apr 25  2003 OpHome.java
-rw-r--r--  1536 Apr 25  2003 Op.java
-rw-r--r--  1536 Apr 25  2003 README
-rw-r--r--  1536 Apr 25  2003 sb.xml
```

Check that all files are present before proceeding to the next section.

5.2. Building the Example

The simplest way to compile this and all examples is to go to the `$JONAS_ROOT/examples/src` directory and enter the command as user `jonas`:

```
ant -find build.xml install
```

5.3. Running the SB Example

Note

This example builds on the example in Section 2.2.2 *Quick Start to the SB Example*.

To run this example, you will have to first start the JOnAS server and then run the Java client. Finally, at the end of the execution you should stop the JOnAS server.

Note

The following example assumes that the current directory is `$JONAS_ROOT/examples/src/sb`.

Here is how to proceed:

1. As `root`, start the Red Hat Application Server server with the following command:
   `/sbin/service jonas start`

2. As user `jonas`, deploy the `sb.jar`:
   `jonas start`

3. Start the Java Client:
   `jclient sb.ClientOp`

   A successful run should output:
   ```
   Create a bean
   Start a first transaction
   First request on the new bean
   Second request on the bean
   Commit the transaction
   Start a second transaction
   Rollback the transaction
   Request outside any transaction
   ClientOp OK. Exiting.
   ```

   Congratulations! You have succeeded running your first EJB application with JOnAS!

4. As `root`, stop the JOnAS server:
   `/sbin/service jonas stop`

5.4. Understanding Session Beans

Here is a description of the files that reside in the `$JONAS_ROOT/examples/src/sb` directory:

- `ClientOp.java` is the Java client that accesses the JOnAS server.
- `jonas-sb.xml` is the JOnAS specific part of the deployment descriptor.
- `OpBean.java` contains the bean implementation code.
- `OpHome.java` contains the home interface of the bean.
- `Op.java` contains the remote interface of the bean.
- `sb.xml` is the standard part of the deployment descriptor.

5.5. Deployment Descriptor

Here is the standard `sb.xml` deployment descriptor:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
    version="2.1">

<ejb-jar>
  <description>Deployment descriptor for the
    sb JOnAS example</description>
  <display-name>sb example</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>Op</ejb-name>
      <home>sb.OpHome</home>
      <remote>sb.Op</remote>
```

```
      <ejb-class>sb.OpBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
      <env-entry>
        <env-entry-name>prop1</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>prop1 value</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>Op</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

Here is the JOnAS specific `jonas-sb.xml` deployment descriptor:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<jonas-ejb-jar xmlns="http://www.objectweb.org/jonas/ns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.objectweb.org/jonas/ns
    http://www.objectweb.org/jonas/ns/jonas-ejb-jar_4_0.xsd">

<jonas-ejb-jar>

  <jonas-session>
    <ejb-name>Op</ejb-name>
    <jndi-name>OpHome</jndi-name>
  </jonas-session>

</jonas-ejb-jar
```

In this example, two beans share the same interface (Account), one uses bean-managed persistence (explicit persistence), the other uses container-managed persistence (implicit persistence). This is a good example of the two persistence techniques.

6.1. Finding the Example Application

The Entity Bean example application is located in the `$JONAS_ROOT/examples/src/eb` directory. It is composed of the following files:

```
-rw-r--r--  19456 Apr 25  2003 AccountExplBean.java
-rw-r--r--   2048 Apr 25  2003 AccountHome.java
-rw-r--r--   1024 Apr 25  2003 Account.idb
-rw-r--r--  10752 Apr 25  2003 AccountImpl2Bean.java
-rw-r--r--  11264 Apr 25  2003 AccountImplBean.java
-rw-r--r--   2048 Apr 25  2003 Account.java
-rw-r--r--   1024 Apr 25  2003 Account.pgsql
-rw-r--r--   1024 Apr 25  2003 Account.sql
-rw-r--r--   6144 Apr 25  2003 ClientAccount.java
-rw-r--r--   9728 Apr 25  2003 db1.prp
-rw-r--r--   4096 Apr 25  2003 eb.xml
-rw-r--r--   2048 Apr 25  2003 jonas-eb.xml
-rw-r--r--   2560 Apr 25  2003 README
```

Check that all files are present before proceeding to the next section.

6.1.1. Understanding Entity Beans

Here is a description of the files that reside in the `$JONAS_ROOT/examples/src/eb` directory:

- `Account.java` contains the Remote interface for the bean Account.

- `AccountExplBean.java` contains the entity bean with "bean-managed persistence".

- `AccountHome.java` contains the Home interface for the bean Account.

- `AccountImpl2Bean.java` contains an entity bean with "container-managed persistence version 2".

- `AccountImplBean.java` contains an entity bean with "container-managed persistence".

- `ClientAccount.java` contains the client account code.

- `jonas-eb.xml` is the JOnAS-specific part of the deployment descriptor.

- `eb.xml` is the standard part of the deployment descriptor.

6.1.2. Building the Example

The simplest way to compile this and all examples is to go to the `$JONAS_ROOT/examples/src` directory and enter the following command line as user `jonas`:

`ant –find build.xml install`

This command compiles all examples in the `$JONAS_ROOT/examples/src` directory.

6.1.3. Configuring Database Access

In order to run this example, you *must* be able to access a relational database. Typically you would use PostgreSQL, as it is preconfigured with Red Hat Application Server. JOnAS will create and use a DataSource object that must be configured for the database you intend to use. These DataSource objects are configured via properties files and are referenced in the `jonas.properties` file (see Section 4.1.2 *JOnAS Configuration Files*).

The `jar` file for your database needs to be in a location where your JRE can find it. (For example, if you are using PostgreSQL and the IBM JRE, the following is necessary:

```
$ su
# cd /usr/lib/jvm/java-java_version-ibm-IBM_java_version/jre/lib/ext
# ln -s /usr/share/java/pg74.215.jdbc3.jar .
```

6.1.3.1. Defining a Datasource

This example requires that a datasource named `jdbc_1` is available in the JOnAS server. This is the default name used in the database properties files provided in the `$JONAS_ROOT/conf` directory. Red Hat Application Server provides configuration files for a variety of common RDBMSs. PostgreSQL is the preconfigured option. `$JONAS_ROOT/conf/PostgreSQL1.properties` is set up to work with the Red Hat Application Server default installation.

6.1.3.2. Creating the Database

JOnAS does not automatically create the database for you—you have to create it prior to running this example. The `Account.sql` file can be used with most SQL-92 compatible databases. There is also an `Account.idb` file that can be used with InstantDB.

For example, for PostgreSQL:

```
$ su
# su -s /bin/sh -c "psql -p 5433 db_jonas" jonas
sampleaccount=# \i Account.pgsql
sampleaccount=# \q
```

Note

The `su` command changes your `pwd`. Thus, after you run `su`, you must go back to the `$JONAS_ROOT/examples/src/eb` examples directory so that the `Account.pgsql` file can be found through the `psql` interface.

Additional information about database configuration is available in the *Configuring Database Service* chapter of the *Red Hat Application Server User Guide*.

6.1.4. Running the EB Example

To do a complete and clean run of this example, you will have to first start the JOnAS server and then run the two Java clients. At the end of the execution, you may stop the JOnAS server.

Note

The following example assumes that the current directory is `$JONAS_ROOT/examples/src/eb`.

Here is how to proceed:

1. As `root`, restart the JOnAS server:
   `/sbin/service jonas start`

2. Load the `eb.jar` file. (See Section 4.1.4 *Loading Beans Using jonas.properties* to Section 4.1.7 *Unloading Beans* for further information.)
   `jonas admin -a eb.jar`

3. Start the two Java Clients:
   `jclient eb.ClientAccount AccountImplHome`
   `jclient eb.ClientAccount AccountExplHome`

   A successful run should output:
   ```
   Getting a UserTransaction object from JNDI
   Connecting to the AccountHome
   Getting the list of existing accounts in database
   101 Antoine de St Exupery 200.0
   102 alexandre dumas fils 100.0
   103 conan doyle 500.0
   104 alfred de musset 100.0
   105 phileas lebegue 350.0
   106 alphonse de lamartine 650.0
   Creating a new Account in database
   Finding an Account by its number in database
   Starting a first transaction, that will be committed
   Starting a second transaction, that will be rolled back
   Getting the new list of accounts in database
   101 Antoine de St Exupery 200.0
   102 alexandre dumas fils 300.0
   103 conan doyle 500.0
   104 alfred de musset 100.0
   105 phileas lebegue 350.0
   106 alphonse de lamartine 650.0
   109 John Smith 100.0
   Removing Account previously created in database
   ClientAccount terminated
   ```

4. As `root`, stop the JOnAS server with the following command:
   `/sbin/service jonas stop`

Note

The `jar` can also be deployed using the web interface. The steps are similar to those described in Section 2.2.2 *Quick Start to the SB Example*, but using the file `eb.jar`.

Message-Driven Beans

Two message-driven beans examples are provided with JOnAS. The first and simplest is located in `$JONAS_ROOT/examples/src/mdb/samplemdb`; it contains a Message Driven bean listening to a topic and an MdbClient, which is a pure JMS Client that sends 10 messages on the corresponding topic. This is a very good example of how to write and use message driven beans.

`$JONAS_ROOT/examples/src/mdb/sampleappli` has the second, more complex, example. The example contains two Message Driven beans—one listening to a topic (StockHandlerBean) and the other listening to a queue (OrderBean), an Entity bean with container managed persistence (Stock-Bean), and a Stateless Session bean for creating the table used in the database.

SampleAppliClient sends several messages on the topic. At receipt of the message, the StockHandlerBean updates the database via the StockBean and sends a message to the Queue inside a global transaction. All the EJBs are involved in transactions that may commit or rollback.

7.1. Building the Examples

The simplest way to compile this and all examples is to start in the `$JONAS_ROOT/examples/src` directory and enter the following command as user `jonas`:

```
ant -find build.xml install
```

This compiles all examples in the `$JONAS_ROOT/examples/src` directory.

7.2. Running the Examples

To run the `samplemdb` example, you will have to first start the JOnAS server and then run the Java client. Finally, at the end of the execution you should stop the JOnAS server.

Note

> `$JONAS_ROOT/examples/src/mdb/samplemdb` is assumed to be the current folder in the following example.

Here is how to proceed:

1. As `root`, start the Red Hat Application Server server with the following command:
   ```
   /sbin/service jonas start
   jonas admin -a samplemdb.jar
   ```

2. As user `jonas`, loar the JAR with the following command:
   ```
   jonas admin -a samplemdb.jar
   ```

3. Start the Java Client:
   ```
   jclient samplemdb.MdbClient
   ```

   A successful run should output something similar to:
   ```
   JMS client: tcf = TCF:localhost-16010
   JMS client: tc = Cnx:#0.0.1030-21
   MDBsample is OK
   ```

4. As `root`, stop the Red Hat Application Server server:
   `/sbin/service jonas stop`

Note

The `jar` can also be deployed using the web interface. The steps are similar to those described in Section 2.2.2 *Quick Start to the SB Example*, but using the file `samplemdb.jar`.

To run the `sampleappli` example, you will have to first start the JOnAS server and then run the Java client. Finally, at the end of the execution you should stop the JOnAS server.

Note

`$JONAS_ROOT/examples/src/mdb/sampleappli` is assumed to be current folder in the following example.

Here is how to proceed:

1. As `root`, start the Red Hat Application Server server with the following command:
   `/sbin/service jonas start`
   `jonas admin -a sampleappli.jar`

2. As user `jonas`, start the Java Client:
   `jclient sampleappli.SampleAppliClient`

   A successful run should output something similar to:
   ```
   For CustomerId = customer1 ProductId= 00001 Quantity= 5
   For CustomerId = customer10 ProductId= 00003 Quantity= 3
   For CustomerId = customer3 ProductId= 00002 Quantity= 2
   For CustomerId = customer2 ProductId= 00003 Quantity= 10
   For CustomerId = customer3 ProductId= 00001 Quantity= 10
   For CustomerId = customer1 ProductId= 00002 Quantity= 5
   For CustomerId = customer122 ProductId= 00003 Quantity= 6
   For CustomerId = customer45 ProductId= 00001 Quantity= 4
   For CustomerId = customer10 ProductId= 00004 Quantity= 3
   For CustomerId = customer7 ProductId= 00002 Quantity= 2
   For CustomerId = customer00 ProductId= 00000 Quantity= 1000
   For CustomerId = customer2 ProductId= 00004 Quantity= 6
   Problem: nb messages sent = 11, nb messages received = 12
           (re-run SampleAppliClient with an argument greater than 10)
   StockId = 00000 Quantity = 10
   StockId = 00003 Quantity = 1
   StockId = 00004 Quantity = 1
   StockId = 00001 Quantity = 1
   StockId = 00002 Quantity = 2
   Problem: Stock id 00002 must be set to 1
           (re-run SampleAppliClent with an argument greater than 10)
   ```

Note

> The "Problems" are part of the expected result. The example starts with some inventory and takes items away as orders comes in; if the example goes into negative stock, it restarts.

   3. As `root`, stop the Red Hat Application Server server:
      /sbin/service jonas stop

Each example comes with a corresponding `README` file.

7.3. Understanding Message-Driven Beans

Here is a description of the files that reside in the `$JONAS_ROOT/examples/src/mdb/samplemdb` directory:

- `README` contains important information that does not appear in the other documentation.
- `run.sh`
- `run.bat`
- `MdbBean.java` contains the message driven bean.
- `MdbClient.java` contains the Client that accesses the Mdb bean.
- `jonas-samplemdb.xml` is the JOnAS specific part of the deployment descriptor.
- `samplemdb.xml` contains the generic deployment descriptor.

The `$JONAS_ROOT/examples/src/mdb/sampleappli` directory contains the following files:

- `README` contains important information that does not appear in the other documentation.
- `run.sh`
- `db1.prp`
- `run.bat`
- `Env.java` contains the remote interface of the bean.
- `EnvBean.java` contains the stateless session bean.
- `EnvHome.java` contains the home interface for the Env bean.
- `jonas-sampleappli.xml` is the JOnAS-specific part of the deployment descriptor.
- `OrderBean.java` contains the message driven bean.
- `sampleappli.xml` contains the generic deployment descriptor.
- `SampleAppliClient.java` is the Java client that accesses the JOnAS server.
- `Stock.java` contains the stock remote interface.
- `StockBean.java` contains the entity bean.
- `StockHandlerBean.java` contains a Message driven bean.
- `StockHome.java` contains the home interface for the stock bean.

Accessing Beans From a Servlet

8.1. Quick Introduction to Servlets

According to *Java Servlet Programming*, 2nd Edition—O'Reilly

> A servlet is a generic server extension—a Java class that can be loaded dynamically to expand the functionality of a server. Servlets are commonly used with web servers, where they can take the place of CGI scripts.

Servlets are executed inside a Java Virtual Machine. The official reference implementation and most commonly used servlet container is Apache's Tomcat Server, which is packaged with Red Hat Application Server and is also freely available from the Apache web site http://jakarta.apache.org.

Servlet/JSP engines are embedded in J2EE™ servers. Red Hat Application Server supports Tomcat.

8.2. Retrieving a Home Interface and Creating a Bean

Here is an example on how to retrieve the home interface of a bean named Region from a Servlet and then create a new instance of this bean:

```
import java.io.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.servlet.*;
import javax.servlet.http.*;

public class RegionServlet extends HttpServlet
{
  /**
   * Called on a GET request for this servlet
   *
   * @param request a <code>HttpServletRequest</code> value
   * @param response a <code>HttpServletResponse</code> value
   * @exception IOException if an error occurs
   * @exception ServletException if an error occurs
   */
  public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException, ServletException
  {
    private PrintWriter out;
    response.setContentType("text/html");
    try
    {
      out = reponse.getWriter();
    }
    catch (IOException ioe)
    {
      ioe.printStackTrace();
    }

    out.println("<!doctype html public
```

```
     "-//w3c//dtd html 4.0 transitional//en">");
   out.println("<html>");
   out.println("<body>");
   out.println("RegionServlet - Looking for the initial context<br>")

   Context initialContext = null;
   try
   {
     initialContext = new InitialContext();
   }
   catch (Exception e)
   {
     out.println("RegionServlet - Cannot get initial context for JNDI: "
     +e+"<br>");
     return ;
   }

   out.println("RegionServlet - Looking up bean's home interface<br>")

   RegionHome home;
   try
   {
     home = (RegionHome)PortableRemoteObject.narrow(
     initialContext.lookup("java:comp/env/ejb/RegionHome"),
     RegionHome.class);
   }
   catch (Exception e)
   {
     out.println("RegionServlet - Cannot lookup home interface: "
     +e+"<br>");
     return ;
   }

   out.println("RegionServlet - Creating a new bean instance<br>")

   Region bean;
   try
   {
     bean = home.create();
   }
   catch (Exception e)
   {
     sp.printHTML("RegionServlet - Error while creating new bean instance: "
     +e+"<br>");
     return ;
   }

   ... here you can call any method defined in the bean
       remote interface (example: bean.doSomething(...);) ...

   out.println("</body>\n");
   out.println("</html>\n");
  }

}
```

8.3. Initiating a Transaction From a Servlet

It is better to initiate the transactions from the beans, however in some cases you want the transaction to be initiated from the servlet. In this case, proceed as follows:

```
...
    try
    {
      Context initialContext = new InitialContext();
      UserTransaction utx = (javax.transaction.UserTransaction)
          initialContext.lookup("java:comp/UserTransaction");
      utx.begin();
      ... do some interesting work here ...
      if (success)
        utx.commit();
      else
        utx.rollback();
    }
    catch (Exception ignore)
    {
      return ;
    }
...
```

Chapter 9.

Accessing Beans From a JSP

The `$JONAS_ROOT/examples/alarm/web/secured/` directory contains JSP samples for the **Alarm** application.

JavaServer Pages (JSP) was Sun's reponse to Microsoft ASP. It looks like a scripting language that basically combines standard HTML and scripting tags. On its first invocation, a JSP is tranlated into a Java servlet. There are many books on this topic if you want to learn more about it.

9.1. Accessing a Bean From a JSP

Accessing a bean from a JSP with JOnAS is very simple. Here is an example skeleton:

```
<html>
<head><title>Test JSP</title></head>
<body>
<jsp:useBean id="myBean" scope="page" class="myPackage.MyBean" />
<%
    myBean.callBusinessMethod();
%>
<p>
</body>
</html>
```

The Alarm Application

Alarm is a simple J2EE™ application developed on top of JOnAS. This application is delivered with the JOnAS distribution and resides in the `$JONAS_ROOT/examples/alarm` directory. This example illustrates how to use JOnAS, Tomcat, and Joram together to create a complete application using EJBs, servlets, html, JSPs, and message driven beans.

This application is run from the "System Management" area. It simulates an administration console receiving alarms from some managed devices. The console is of course an HTML client visualizing the alarms received and maintained in the database by the application server.

10.1. Application Architecture Overview

This application shows how to use JOnAS, Tomcat, and Joram together in a complete application using EJBs, servlets, html, JSPs, and message driven beans. Moreover, a JOnAS "service" is run inside the J2EE server. It is composed of the following parts:

- AlarmGenerator: AlarmGenerator is a servlet that generates alarms by publishing messages on a topic. Once the JOnAS Server is running, this servlet may be used to feed the Message Driven Bean listening on the topic.

- Session bean View: A Session bean is used to provide a remote access to the Alarm Service. Each user will create its own session that will be reached from JSP pages or servlets.

- Entity Bean AlarmRecord: When a new alarm type is received, an entity bean is created. If the Alarm is already known, its count is incremented only.

- Message Driven Bean: JMS is used to access asynchronously to the service. A Message Driven Bean is used to collect Alarm sent by AlarmGenerator. It will then inform the AlarmManager that will process it.
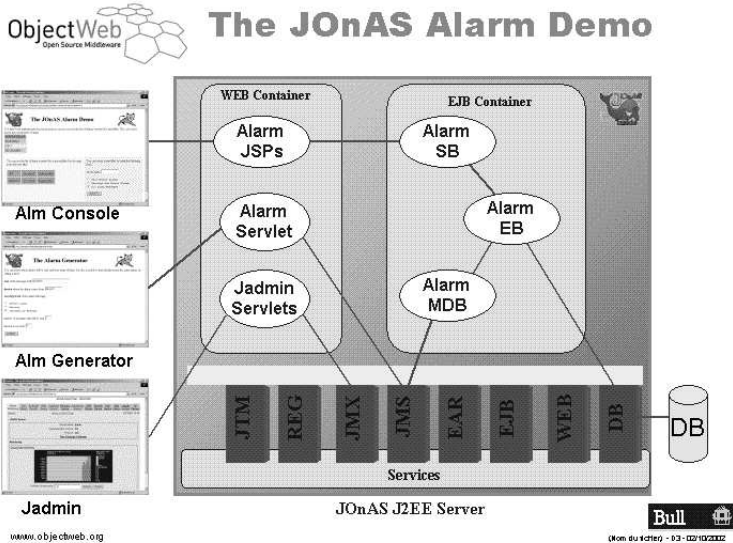
Figure 10-1. Alarm application architecture overview

10.2. Finding the Alarm Application

The Alarm application is located in the `$JONAS_ROOT/examples/alarm` directory. It is structured as follows:

- `etc/web.xml`: application description.
- `web/` directory: JSPs, images, html files.
- `beans/` directory: beans and service to be loaded in JOnAS.
- `src/` directory: servlet generating alarms.
- `client/` directory: client java program generating alarms and using JMS, now obsolete; replaced by the servlet in `src/`.

10.3. Setting Up the Application

You should have installed Tomcat using the combined JOnAS and Tomcat package as explained in this tutorial.

The first step is to compile the alarm application. The simplest way is to go to the `$JONAS_ROOT/examples/alarm` directory and enter the following command as user `jonas`:

```
ant –find build.xml compile
```

Note

Make sure that the `bcel.jar` file is in the `/usr/share/java/` directory.

10.4. Configuring Database Access

You must have configured a Database. The jndi name used in the bean is `jdbc_1`. See Section 4.1.3 *Database Access* in this tutorial or the *Configuring Database Service* section of the *Red Hat Application Server User Guide*.

You do not have to create the database (table), it will be done by the application (the first time you launch the JOnAS server you will see an exception saying that the table does not exist; do not worry about this—it is a normal behaviour—the table will be created just after).

10.5. Running the Alarm Demo

To run the alarm example, you will have to first start the JOnAS server and then run **jonas admin**. Finally, at the end of the execution you should stop the JOnAS server.

Note

The following example assumes that the current folder is `$JONAS_ROOT/examples/alarm`.

Here is how to proceed:

1. As `root`, start the Red Hat Application Server server with the following command:
   /sbin/service jonas start

2. Run `jonas admin` with the following arguments:
   jonas admin -a alarm.ear

   Note

   The EAR can also be deployed using the web interface:

   a. Click **Deployments > Applications (EAR)**.
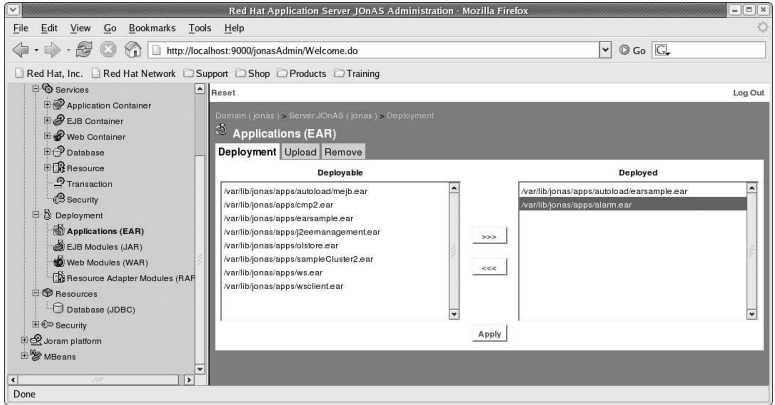
Figure 10-2. Deploy the alarm.ear Example

b. On the `alarm.ear` in the **Deployable** list box on the left side, Click **Deploy**, then click **Apply**.

c. A **Confirm** dialog appears; click **Confirm**.
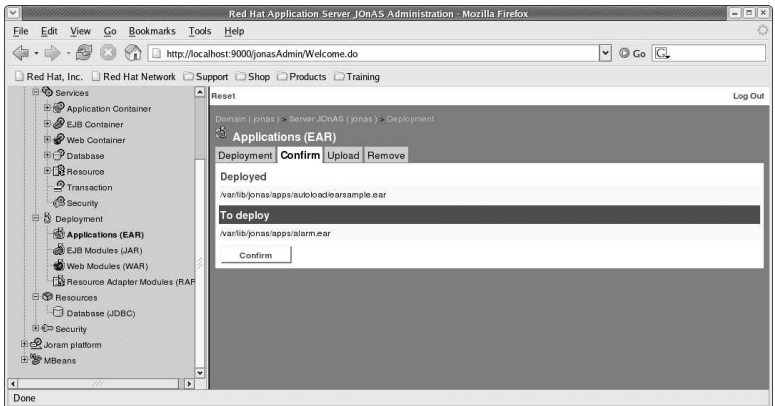
Figure 10-3. Confirm the alarm.ear Deployment

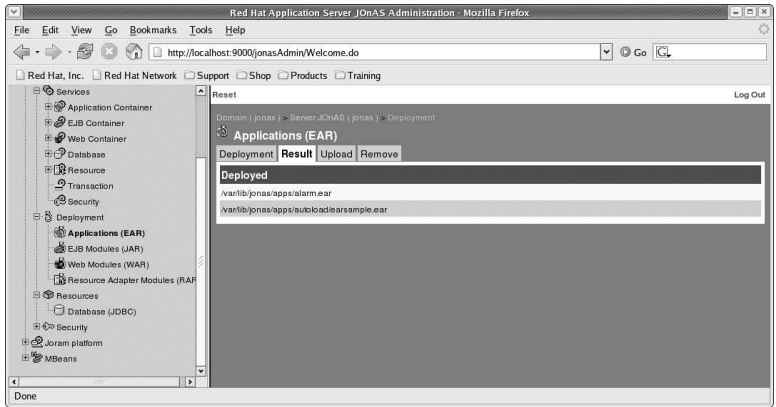d. A **Result** dialog appears, and the newly deployed `EAR` is in the list.

Figure 10-4. Results of the alarm.ear Deployment

    e. If you click **Deployments > Applications (EAR)** again, `alarm.ear` will appear in the right list box, which lists the deployed `EAR` files.

If you click **Application Container**, **EJB container**, or **Web Container** under **Services**, you will see `alarm.ear` in the tree.

3. Generate alarms: The easiest way is to use the link on the first html page that will run a servlet that will generate customized alarms. Look at http://$HOSTNAME:9000/alarm/index.html to start using the web application. If you are asked for authentication, use **Username: `tomcat`** **Password: `tomcat`**.

Running **Alarm** is simply a matter of following the on-screen instructions: to set alarms, click **Click here to generate alarms**; to check for alarms, click **Filters**.

4. As `root`, stop the Red Hat Application Server server:
   `/sbin/service jonas stop`

10.6. Known Bugs or Limitations

- The first time you run this application, the table will be created in the database. An error message will be printed by the J2EE server because it tries to drop the table, but the table does not exist. Ignore it.
- The display pages are not automatically refreshed when a new alarm is received. To be done in a future version.
- Filter names cannot contain "space" characters.
- Sending many alarm messages concurrently with InstantDB as the database will not work (a deadlock will lead to a transaction timeout, and a delay on servlet display). This can occur with every database that cannot handle multithreading properly.

Chapter 11.
Documentation

This chapter describes additional sources of information about Red Hat Application Server.

11.1. Documentation

For more information about JOnAS, please read the documentation that comes with the Red Hat Application Server distribution.

Note that the latest release JOnAS documentation can always be found online.

11.1.1. Release Documentation

The Red Hat Application Server documentation consists of this tutorial, the *Installation Guide*, which explains how to install Red Hat Application Server, and the *Red Hat Application Server User Guide*, which contains the following sections:

- *Configuration Guide* explains how to configure JOnAS. You will learn how to deal with the JOnAS configuration files in order to set up the JOnAS and the communication environment, the registry launch, and the JOnAS services (EJB, Web Container, EAR, JDBC, Security, Transaction, Messaging, Management, JCA resources, and Mail).

- *J2EE Application Programmer's Guide* shows in great details how to develop Session, Entity, or Message-Driven Beans with JOnAS. It shows in particular how to manage the security and transactional behavior, how to configure JDBC datasources, how to define the EJB deployment descriptors, and how to package the written beans.

- *JOnAS Commands Reference Guide* is a reference guide to the JOnAS commands: `newbean`, `registry`, `CheckEnv`, `jonas`, `GenIC`, `JmsServer`, `jonasAdmin`, and `RAConfig`.

- http://jonas.objectweb.org/current/doc/howto/JORAMdistributed.html   *JOnAS and JORAM: Distributed Message Beans*

- http://jonas.objectweb.org/current/doc/howto/WebSphereMQ.html *Howto: Using WebSphere MQ JMS Guide*

- http://jonas.objectweb.org/current/doc/Config.html#Config-JDBC-RAs   *Configuring JDBC Resource Adapters*

11.2. Mailing Lists

Two mailing lists about JOnAS are hosted by the ObjectWeb Consortium: jonas@objectweb.org and jonas-team@objectweb.org.

11.2.1. JOnAS Users Mailing List

The jonas@objectweb.org mailing list is the general public mailing list for JOnAS users. Use this list for any question about JOnAS, problems, bug reporting, announcements, contributions, etc. Please *read first* the JOnAS documentation and the archives before posting a question. The JOnAS community typically responds quickly to your questions.

The traffic of this mailing list is moderate. Don't hesitate to register in order to be informed of the latest JOnAS news. To do this, simply send a blank e-mail to: jonas-subscribe@objectweb.org. You can also use the online Sympa interface (Sympa is the mailing list software management used by ObjectWeb). To browse and search the archives, go to the jonas@objectweb.org mailing list page.

11.2.2. JOnAS Team Mailing List

The jonas-team@objectweb.org is the mailing list you can use if you want to directly contact the JOnAS project members. For general questions about JOnAS, please use the jonas@objectweb.org public mailing list.

A

Ant

Ant, from Apache, is a Java-based build tool (see http://ant.apache.org/).

B

BCEL (Byte Code Engineering Library)

The BCEL package, from Apache, is required for correct operation of the Ant build scripts used by JOnAS. BCEL gives users a convenient way to analyze, create, and manipulate (binary) Java `.class` files. Classes are represented by objects that contain all the symbolic information of the given class: methods, fields, and byte code instructions, in particular.

BMP (Bean-Managed Persistence)

Bean-managed persistence (BMP) occurs when the entity object manages its own persistence. The enterprise bean developer must implement persistence operations (such as JDBC) in the enterprise bean class methods.

C

CAROL (Common Architecture for RMI Objectweb Layer)

CAROL (Common Architecture for RMI Objectweb Layer) is a library that enables the use of different RMI implementations (see http://carol.objectweb.org). With CAROL, a Java server application can be independent of RMI implementations and accessible simultaneously by RMI clients using different RMI implementations. CAROL allows the design, implementation, compilation, packaging, deployment, and execution of distributed applications compliant with the RMI model.

CMI (Cluster Method Invocation)

A new ORB used by JOnAS to provide clustering for load balancing and high availability.

CMP (Container-Managed Persistence)

CMP can be used to isolate an application developer from the physical database schema. This is done by creating an abstract schema that matches the needs of the current application, and using CMP to map between the abstract and physical schemas. The use of CMP ensures that applications are portable in the event of vendor or schema changes.

CMP is defined by the Enterprise JavaBeans 2.0 specification (see http://java.sun.com/products/ejb/2.0.html).

CORBA (Common Object Request Broker Architecture)

CORBA is Object Management Group's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program can interoperate with any other CORBA-based program. See http://www.omg.org/gettingstarted/corbafaq.htm.

E

EAR service

A service used for deploying complete J2EE applications; that is, applications packaged in EAR files, which themselves contain ejb-jar files and/or WAR files.

EJB (Enterprise JavaBean)

The Sun Enterprise JavaBeans specification defines an architecture and interfaces for developing and deploying distributed Java server applications based on a multi-tier architecture. See http://java.sun.com/products/ejb/docs.html.

The intent of this specification is to facilitate and normalize the development, deployment, and assembly of application components (called enterprise beans); such components will be deployable on EJB platforms. The resulting applications are typically transactional, database-oriented, multi-user, secured, scalable, and portable. More precisely, this EJB specification addresses the runtime environment, called the EJB server, which provides the execution environment together with the transactional service, the distribution mechanisms, the persistence management, and the security.

I

IIOP (Internet Inter-ORB Protocol)

IIOP maps GIOP messages to TCP/IP. The GIOP (General Inter-ORB Protocol) is a collection of message requests that ORBs can make over a network.

J

J2EE CA (J2EE Connector Architecture)

J2EE Connector Architecture. Sun's J2EE Connector Architecture provides a Java solution to the problem of connectivity between the many application servers and today's enterprise information systems.

JAAS (Java Authentication and Authorization Service)

JAAS is a set of APIs that enable services to authenticate and enforce user-based authorization.

JACC (Java Authorization Contract for Containers)

JOnAS implements the Java Authorization Contract for Containers (JACC 1.0) specification. This enables you to manage authorizations as Java security permissions, and to plug in any security policy provider.

JDBC (Java DataBase Connectivity)

JDBC is an API that provides cross-database connectivity and access to other data sources, such as spreadsheets or flat files. See http://java.sun.com/products/jdbc.

JDO (Java Data Objects)

JDO defines interfaces and classes to be used by application programmers when using classes whose instances are to be stored in persistent storage (persistence-capable classes), and specifies the contracts between suppliers of persistence-capable classes and the runtime environment (which is part of the JDO Implementation).

Jeremie

Jeremie is the RMI personality of Jonathan. It can be used in JOnAS to replace the standard RMI implementation and to optimize local calls.

Jetty

Jetty is a Java HTTP Server and Servlet Container, which means that you do not need to configure and run a separate web server (such as Apache) in order to use java, servlets, and JSPs to generate dynamic content.

JMS (Java Messaging Service)

The JMS API enables J2EE application components to create, send, receive, and read messages.

JMS is provided by JORAM, a technology from ScalAgent (http://www.scalagent.com). See http://java.sun.com/products/jms.

JMX (Java Management Extension)

JOnAS contains the Java Management Extension technology. Management and monitoring is available through a Web interface. See http://java.sun.com/products/JavaManagement.

JNDI (Java Naming and Directory Interface)

JNDI provides Java applications with a unified interface to naming and directory services. See http://java.sun.com/products/jndi.

JOnAS

JOnAS (Java Open Application Server) is a distributed platform compliant with the EJB^{TM} specifications (see http://www.objectweb.org/jonas/index.html). JOnAS provides object distribution, security, distributed transactions, and object-persistence support according to these specifications. JOnAS also provides JMS^{TM} (Java Message Service) and JCEE^{TM} CA (J2EE Connector Architecture) support through its integration with JORAM and JORM.

Jonathan

Jonathan is an adaptable, distributed-object platform, that currently provides several personalities, including one compliant with the OMG CORBA specifications and another one with the RMI specification (see http://www.objectweb.org/jonathan/index.html). The RMI personality, called Jeremie, can be used in JOnAS to replace the standard RMI and optimize local calls.

JORAM (Java Open Reliable Asynchronous Messaging)

JORAM is an open-source implementation of the JMS (Java Message Service) specification (see http://www.objectweb.org/joram/index.html). JORAM provides a MOM (Message Oriented Middleware) built on top of the ScalAgent distributed-agent technology (see http://www.scalagent.com).

JORM (Java Object Repository Mapping)

JORM is an adaptable persistence service that offers various personalities, including one compliant with the CMP EJB specification and another with the JDO (Java Data Objects) specification (see http://www.objectweb.org/jorm/index.html). JORM provides object persistency through different secondary storage support, such as file, relational databases, or object-oriented databases. JORM includes an implementation of the JCA (Java Connector Architecture) specifications.

JavaServer Pages (JSP)

JSP provides a fast, easy way to create server- and platform-independent web applications (see http://java.sun.com/products/jsp/).

JTA (Java Transaction API)

JTA specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.

JVM (Java Virtual Machine)

A JVM is a kind of translator that turns general Java platform instructions into tailored commands for devices.

M

MOM (Message Oriented Middleware)

MOM is a specific class of middleware that supports the exchange of general-purpose messages in a distributed application environment. Messages may be synchronous or asynchronous. The MOM system provides the directory, security, and administrative services required to support messaging.

In JOnAS, JORAM provides a MOM built on top of the ScalAgent distributed-agent technology (see http://www.scalagent.com).

O

ORB (Object Request Broker)

The Object Request Broker (ORB) is the component of CORBA that provides all the communication infrastructure needed to identify and locate objects, handle connection management, deliver data, and request communications.

R

RMI/IIOP (Java Remote Method Invocation over Internet Inter-Orb Protocol)

RMI/IIOP delivers CORBA distributed computing capabilities to the Java 2 platform. JOnAS now provides support for RMI/IIOP by integrating the CAROL communication framework.

S

Servlet

Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing business systems. A servlet is like an applet that runs on the server side—but without a face.

T

Tomcat

A Web Container Service is a servlet/JSP engine in the JVM of the Red Hat Application Server server and the loading of web applications ("WAR files") within this engine. Currently this service can be configured to use Tomcat or Jetty, although only Tomcat is supplied with Red Hat Application Server.

W

WAR files

Web application files.

Web server

The Web server is responsible for accepting client requests and sending HTML replies back to the client. HTML pages can be static files that reside on the Web server filesystem or dynamically built with Servlets or JSPs from data generated by the beans.

X

XA (Distributed transaction mode for JDBC 2.0)

The JDBC Standard Extension API, part of Sun's Enterprise JavaBeans (EJB) technology, enables you to write distributed transactions that use connection pooling. It also makes it possible to connect to virtually any tabular data source, including files and spread sheets.

Index