



---

# **Response Tester**

## ***Installation Guide***

---

**Author:** Jon Barnett

**Document version:** 2.0

# Table of Contents

1-Introduction.....	1
2-Software description.....	2
3-Environmental pre-requisites.....	3
3.1-Hardware platform.....	3
3.1.1-Application specific requirements.....	3
3.2-Application servers.....	3
3.2.1-Servlet servers.....	3
3.2.2-EJB servers.....	3
3.3-Databases.....	3
4-Installation for JBoss-Tomcat.....	4
4.1-Installing the Java Development Kit.....	4
4.2-Installing JBoss 3.2.x.....	4
4.3-Defining the run-time environment variables.....	4
4.4-Amity support components.....	5
4.5-Connection pool configuration.....	6
4.6-Ports and interfaces used by JBoss.....	7
4.7-Trial the basic installation.....	8
4.8-Amity Enterprise JavaBean deployment.....	8
4.9-Web application deployment.....	9
4.10-Statistics collection.....	9
4.11-Statistics display.....	9
4.12-Linux boot scripts.....	9
4.12.1-Pre-requisites.....	9
4.12.2-Red Hat script.....	10
4.12.3-Slackware script.....	10
4.12.4-Post installation.....	10
5-Database setup.....	11
5.1-Response storage.....	11
5.2-Event logging.....	11
5.3-Statistics collection.....	12
5.4-Database access.....	12
6-JBoss 3.2.x recommendations.....	13
6.1-Connection pool.....	13
6.2-Web application server tuning.....	13
6.2.1-Tomcat.....	13
6.2.2-Jetty.....	14
6.2.3-JSP tuning.....	14
6.3-Security.....	18
6.3.1-JMX-console and Web-console.....	18
6.4-Unused services.....	21
6.5-Logging.....	21
7-Standalone Tomcat configuration.....	23
7.1-Tomcat 4.1.....	23
7.1.1-Amity support components.....	23
7.1.2-Support components.....	23
7.1.3-Deployment modifications.....	24
7.1.4-Response tester package deployment.....	27
7.1.5-Special considerations.....	27
7.1.6-Alternative modifications.....	28
8-Troubleshooting for JBoss.....	34
8.1-JBoss/Tomcat configuration checklist.....	34
8.2-Specific installation cases.....	34
8.3-Operational troubleshooting.....	35
8.3.1-Unable to write to the database.....	35
8.3.2-Unable to find the writer service.....	35
8.3.3-JSP pages don't display after some time.....	36
8.3.4-JBoss performs a clean shutdown by itself.....	36

# 1 Introduction

The purpose of this document is to describe the installation process for the response testing software. It covers the environmental prerequisites for the system and describes some of the assumptions that must be verified for the purposes of accurate measurement.

This guide also provides in-depth instructions for installing and configuring the response tester software for the JBoss 3.2.x range of Java 2 Enterprise Edition (J2EE) compliant application servers. It provides an update to the original software and documentation in covering installation and run-time requirements for the latest production releases of JBoss. These releases also include an embedded Tomcat or Jetty web application server.

The response tester software has minimal changes other than to take advantage of EJB 2.0 interfaces for performance reasons. The deployment descriptors have also been modified to take advantage of the new J2EE environment. These changes have no impact on the logical functionality of the software.

## 2 Software description

The response testing software generates a test page with an arbitrary amount of content, definable by the test engineer. Upon completion of the page loading at the user's browser, a redirect command will load the result page. The system measures the time between the request for the test page and the result page. Since the sizes of the requests are comparatively small compared to the size of the test page, the response time can be reasonably approximated as the time to request and load the test page.

The test engine is configured to save results to a database on user request but this may be reconfigured by the test engineer to automatically save the response times rather than requiring user input. Such work requires knowledge of JSP programming and is not covered in this guide. The implementation is well encapsulated and the proficient developer can quickly develop new JSPs that use the Amity components, using the application JSPs as a starting point. The database access service is scalable and dependent on the EJB server capabilities to provide connection pools. The EJB servers recommended by Amity Solutions provide these services.

Statistical calculations are performed as a separate scheduled task. The calculations provide a frequency analysis of measurement samples as well as determining the average and standard deviation for each sample set. The calculated results are suitable for display in a frequency histogram. Sample graphing is provided for visual analysis of results and this is given as an example in the supplied JSP code. The access to this graphing service may be redefined by the test engineer – for example, as a separate page or a page accessed only by specific users.

The software is built on a J2EE framework and therefore supports scalability, fast deployment and robustness. It is 100% Java and will run on any application server platform capable of supporting a Java 2.0 VM (Virtual Machine) where the JDK version is greater than or equal to 1.3.1. The design and tier separation ensures that the database is protected from heavy loading and ensures that services adapt to load within the bounds of the hardware platform.

In order to run, the software requires a Java Servlet server and an EJB (Enterprise Java Bean) server. Commercial server products are available from IBM, BEA, Novell and Sun. However, Amity products will also run on open server products that support the relevant J2EE standards.

## **3 Environmental pre-requisites**

### **3.1 Hardware platform**

The system will run on any platform environment that supports JDK 1.3 or higher and additionally the platform environment supports the application servers (servlet engine and EJB server).

#### **3.1.1 Application specific requirements**

**Java Virtual Machine (JVM):** ≥ Java Development Kit (JDK) 1.3.1

**Nominal disk space:** 10 Mb

**Nominal memory requirements:** As necessary to run the application servers

### **3.2 Application servers**

The application servers must be installed before deploying the response tester system components. Any application servers compliant with Servlet specification 1.3, JSP specification 1.2 and EJB specification 2.3 may be employed. Tested J2EE application servers are listed below.

#### **3.2.1 Servlet servers**

- BEA WebLogic
- IBM WebSphere
- Novell SilverStream server
- ≥ Apache Tomcat 4.03
- ≥ Jetty 4.2.10

#### **3.2.2 EJB servers**

- BEA WebLogic
- IBM WebSphere
- Novell SilverStream server
- ≥ JBoss 3.2.0

### **3.3 Databases**

The system can support the following databases using the relevant JDBC adapters:

- ≥ Oracle 8i
- ≥ DB2 UDB 7.2
- ≥ Postgresql 2.3
- Microsoft SQL Server
- Microsoft Access

## 4 Installation for JBoss-Tomcat

The following installation instructions apply specifically to an integrated JBoss-Tomcat application server environment. Refer to the documentation supplied with other J2EE application servers to determine the placement and configuration of components.

### 4.1 Installing the Java Development Kit

Ensure the Java Development Kit is installed and operational. The Java Development Kit is required as the Tomcat system will require the Java compiler toolkit to compile Java Server Pages (JSP). In a Microsoft environment, do not install the JDK in a directory path containing a space.

### 4.2 Installing JBoss 3.2.x

Unpack and install the JBoss 3.2.x – Tomcat package. The package may be obtained from the JBoss website, <http://www.jboss.org/>. Simply unpack the package into the target installation directory.

For a UNIX/Linux installation, we would recommend either unpacking the contents of the package into /opt/JBoss-version or /usr/local/JBoss-version, using the appropriate version number for your JBoss installation. The latest version is JBoss 3.2.2 so the installation directory would be /usr/local/JBoss-3.2.2.

Similarly, you should install JBoss in an appropriate directory on your Microsoft server. You should not install your application server in a directory path that contains a space.

From now on, we will refer to the JBoss root directory as JBOSS\_HOME. Taking our previous example, this would mean that JBOSS\_HOME=/usr/local/JBoss-3.2.2.

The sub-directories under the installation root should be **bin**, **client**, **docs**, **lib** and **server**. The bin sub-directory contains the JBoss startup and shutdown scripts and Java libraries for these operations. It will also contain the Operating System (OS) environmental properties for the correct start operation of the commands. The client sub-directory has various support library configurations for remote clients to connect to JBoss and use EJB resources. The docs sub-directory contains Document Type Definitions (DTD) for deployment descriptors, example configuration files for JBoss-supported services including Java Connector Architectures (JCA) such as Java Database Connection (JDBC) based datasources. The lib sub-directory contains global and bootstrap libraries necessary for the JBoss microkernel before the remainder of JBoss service modules are loaded. The server sub-directory contains various service run-time configurations for JBoss. The standard run-time configurations provided are all, default and minimal. These configurations are self-explanatory.

### 4.3 Defining the run-time environment variables

JBoss requires a few environment variables defined for error-free operation. JAVA\_HOME should refer to the root directory of the JDK installation. JBOSS\_CLASSPATH should be clear, particularly of any J2EE libraries such as servlet.jar or j2ee.jar as JBoss provides all these infrastructure components.

You can modify the JAVA\_HOME definition in the JBOSS\_HOME/bin/run.conf. The run.sh and run.bat scripts used to start JBoss depending on your OS, will read this configuration file and use any definitions contained within it. So if the JDK was installed in /usr/local/IBMJava2-141, the **run.conf** fragment would be:

```
#
# Specify the location of the Java home directory. If set then $JAVA will
# be defined to $JAVA_HOME/bin/java, else $JAVA will be "java".
#
JAVA_HOME="/usr/local/IBMJava2-141"
```

The JBoss run scripts use `JBOSS_CLASSPATH` rather than `CLASSPATH` directly, in order to minimise `CLASSPATH` contamination for the JBoss JVM. You can either create a definition for `JBOSS_CLASSPATH` or `JBOSS_BOOT_CLASSPATH` in `run.conf` containing any additional libraries or classes you wish to include in your Jboss-Tomcat JVM. Your other option is to place these libraries in your J2EE deployment packages or in the JBoss run-time library directories `JBOSS_HOME/lib` or `JBOSS_HOME/server/instance/lib` where `instance` is the run-time instance of JBoss you will be using. The latter option is the more usual approach to including additional libraries. An example of a definition in `run.conf` is:

```
#
# Specify the location of any special libraries or classes for your J2EE
# applications.
#
JBOSS_BOOT_CLASSPATH="/usr/local/XML/lib/crimson.jar"
```

This is not required for the response tester software as no additional libraries are required to be defined external to the JBoss environment.

## 4.4 Amity support components

These are shared components used by the Amity EJBs and the web application. Consult the documentation supplied with your application server to determine where shared libraries are to be placed in order to be loaded by your application server.

For the JBoss installation, we will be using the default configuration. For JBoss, the following Amity support components must be installed in the `JBOSS_HOME/server/default/lib` directory:

- `corporatecore.jar`
- `ejbcore.jar`
- `enhancedejbcore.jar`
- `flexweb.jar`

There are also two files required that will vary according to the database you are using to store results. These files will also be placed in the `JBOSS_HOME/server/default/lib` directory .

The `dataejbcore.jar` file provides base functionality for the EJB management.

- `postgresql.dataejbcore.jar` (if running PostgreSQL)
- `oracle.dataejbcore.jar` (if running Oracle)
- `db2.dataejbcore.jar` (if running DB2 UDB 7.2)
- `access.dataejbcore.jar` (if running Access or SQL Server)

The `sqlmanager.jar` file provides base functionality for the sql query engine.

- `access.sqlmanager.jar` (if running Access or SQL Server)
- `oracle.sqlmanager.jar` (if running Oracle)
- `db2.sqlmanager.jar` (if running DB2 UDB 7.2)
- `postgresql.sqlmanager.jar` (if running PostgreSQL)

You will also need to copy two third-party Java libraries into `JBOSS_HOME/server/default/lib`. These libraries are **trove.jar** and **JOpenChart.jar**. These files are located in the `/Servers` directory of the response tester distribution.

The JDBC driver for the database you are going to use must also be installed in the directory, `JBOSS_HOME/server/default/lib`:

- `postgresql.jar` (if running PostgreSQL)
- `classes12.zip` (if running Oracle 8i) or an appropriate 3<sup>rd</sup> party driver
- `db2java.zip` (if running DB2 UDB 7.2) or an appropriate 3<sup>rd</sup> party driver
- `jtds-0.5.jar` (if running SQL Server) or an appropriate 3<sup>rd</sup> party driver
- any appropriate 3<sup>rd</sup> party driver (if running Microsoft Access)

The JDBC drivers for commercial databases are provided with the DataBase Management System (DBMS) distributions. Due to licensing and version compatibility issues, these drivers should be obtained from your database vendor.

## 4.5 Connection pool configuration

JBoss 3.2.x delivers a simplified connection pool configuration compared to previous releases. Copy the appropriate connection pool template for your database from `JBOSS_HOME/docs/examples/jca` into `JBOSS_HOME/server/default/deploy`. The suitable templates are:

- `postgres-ds.xml` for PostgreSQL
- `oracle-ds.xml` for Oracle
- `db2-ds.xml` for DB2
- `mssql-ds.xml` for SQL Server
- `msaccess-ds.xml` for Microsoft Access

You can rename the template anything you wish, but the file name must end with **-ds.xml** so that JBoss recognises that the configuration represents a datasource.

The connection pool requires a login and password for the database with sufficient access to read and write to the database. The relevant tags in the datasource configuration are:

```
<user-name>x</user-name>
<password>y</password>
```

The name of the connection pool must be `AmityPool` as this is name used by the Amity deployment descriptors. The modified tag in the datasource configuration is:

```
<jndi-name>AmityPool</jndi-name>
```

Ensure that your JDBC driver as specified in the datasource configuration matches the JDBC driver you are using. This is specified by the `driver-class` tag in the datasource XML.

```
<driver-class>org.postgresql.Driver</driver-class>
```

Modify the entry for the `connection-url` tag to match the required JDBC URL for your database. For example:

```
<connection-url>jdbc:postgresql://myserver:5432/Amity</connection-url>
```

The JDBC URL defined in the `connection-url` element will need to be replaced as per the URL requirements of the JDBC driver you employ. Refer to your JDBC driver documentation for this information. Typically for the more commonly used databases, these will be of the form:

- `jdbc:postgresql://address:port/database` for PostgreSQL
- `jdbc:oracle:thin:@address:port/database` for Oracle 8i or higher
- `jdbc:db2://address:port/database` for DB2 UDB 7.2 or higher
- `jdbc:microsoft:sqlserver://address:port;DatabaseName=database` for SQL Server

## 4.6 Ports and interfaces used by JBoss

JBoss 3.2.x contains an embedded web application container – either Tomcat or Jetty. By default, these are configured to serve web content on port 8080 across all network interfaces recognised by the OS. You will obtain errors for the web application container if there is any other service bound to port 8080. Ensure there are no other applications using this port. There are a number of other ports used by JBoss.

- 1098 RMI naming service (JBoss 3.2.2 and later)
- 1099 JNP naming service (JNDI)
- 1701 Hypersonic database service (JBoss 3.2.1 and earlier)
- 4444 RMI object service
- 4445 Pooled invoker
- 8009 Embedded servlet container JK listener
- 8080 Embedded servlet container listener (default HTTP service)
- 8083 Web service for deployment classloading
- 8090 JMS OIL service
- 8091 JMS UIL service (JBoss 3.2.1 and earlier)
- 8092 JMS OIL2 service
- 8093 JMS UIL2 service
- 8443 Embedded servlet container SSL listener (default HTTPS service)

The Java Messaging Service (JMS) related port bindings can be changed by editing the relevant configuration files located in `JBOSS_HOME/server/default/deploy/jms` for JBoss 3.2.2 and later, otherwise editing the configuration file `JBOSS_HOME/server/default/deploy/jbossmq-service.xml` in JBoss 3.2.1 and earlier.

The Hypersonic database should normally be configured for no port binding unless there are external systems that need to connect to it. The non-port bound configuration is the default for JBoss 3.2.2 and later. You can edit `JBOSS_HOME/server/default/deploy/hsqldb-ds.xml` to accomplish this in earlier JBoss 3.2.x releases. This involves commenting out the MBean entry:

```
<!-- This mbean should be used only when using tcp connections. Uncomment
when the tcp based connection-url is used.
<mbean code="org.jboss.jdbc.HypersonicDatabase"
  name="jboss:service=Hypersonic">
  <attribute name="Port">1701</attribute>
  <attribute name="Silent">true</attribute>
  <attribute name="Database">default</attribute>
  <attribute name="Trace">>false</attribute>
  <attribute name="No_system_exit">true</attribute>
</mbean>
-->
```

The connection URL must be configured for a destination directory (JBoss 3.2.2 and later):

```
<connection-url>jdbc:hsqldb:${jboss.server.data.dir}/hypersonic/localDB
</connection-url>
```

JBoss 3.2.1 and earlier will most likely require the connection URL defined as:

```
<!--connection-url>jdbc:hsqldb:./</connection-url-->
```

The temporary file will be created in the JBOSS\_HOME/bin directory.

The remainder of the port binding configurations can be located in JBOSS\_HOME/server/default/conf/jboss-service.xml. Modify the ports as necessary for your environment to ensure there are no clashes with existing services. However, the JBoss defaults should be retained wherever possible to allow for easier upgrade paths for future JBoss releases.

JBoss 3.2.x has these services bound across all network interfaces provided by the OS. The binding is performed on 0.0.0.0 by the JVM. In JBoss 3.2.2, the binding can be overridden on a service level by replacing the `{jboss.bind.address}` value with a hard-coded value. However, we would recommend that this is not done without consultation with Amity Solutions or JBoss Group as the changes could have ramifications on the correct operation of the JBoss application server.

## 4.7 Trial the basic installation

Trial the installation by executing the 'run' script – either **run.bat** in a Microsoft environment or **run.sh** in a Unix environment. The script is located in JBOSS\_HOME/bin and you must run these scripts from that directory. Running these scripts without any command line arguments will result in the default run-time configuration being used.

JBoss should start without any errors being generated in the logs and the final line should be a summary of the length of time for JBoss to start. The logs can be located in JBOSS\_HOME/server/default/log. The **server.log** contains the majority of the information related to services and will be the default one to which we refer in this guide. The **boot.log** contains information on the bootstrap of the JBoss microkernel. This can usually be ignored unless JBoss fails to start at all. The log information for the JBoss start up is also echoed to the console. However, errors are sometimes difficult to trace via the console for fast systems.

Should you find that the server.log indicates some errors, you may find that the connection pool could not establish connections to the database so confirm your settings. Refer to the JBoss documentation for more information on connection pools and installing JDBC drivers if necessary, although the information provided in this guide should be sufficient.

Other sources of problems may be port binding clashes as described in Section 4.6. Investigate and resolve these port bindings as necessary.

## 4.8 Amity Enterprise JavaBean deployment

A single Amity Enterprise JavaBean (EJB) exists for controlling access to the measurement data. JBoss has a hot deployment service that allows you to deploy new active components while the service is running. With JBoss running, place the Amity component, **responses.jar** in the JBOSS\_HOME/server/default/deploy directory. There is another bean **eventlogger.jar** that must also be deployed. It logs any problems that may occur with the operation of the main EJB. Finally, you may install an optional statistics collection service contained in the Amity component, **frequencies.jar**. Refer to section 4.10 for more information on the statistics collection.

Check the JBoss logs and ensure no errors were generated and that the beans were correctly deployed by the server.

This completes the EJB service installation.

## 4.9 Web application deployment

The web interface for the response tester is contained in a Web ARchive (WAR). This is a J2EE component that is deployed in a similar manner to the EJB components. With JBoss running, place the Amity component, **responsetester.war** in the `JBOSS_HOME/server/default/deploy` directory.

Check the JBoss logs and ensure no errors were generated and that the web application was correctly deployed by the server.

This completes the WAR installation.

## 4.10 Statistics collection

There is a separate module to compile frequency data for the graphs of responses. The extraction and use of the data can be determined from the sample JSP.

Since this is not necessary as a part of the measurement service, it is contained in a separate distribution tar ball package called `Collector.tar.gz`. Unpack all the files into the directory in which you want the service to run.

The statistics collection task is intended to be run by cron or a similar scheduler. A Unix shell script is provided with the distribution as an example. **StatsCollector.sh** must be modified to run with your environment, including changes you may have made with the address and port the JBoss directory service uses. The script is straightforward and can be modified by the Unix administrator. Create the scheduling to match your update requirements. A 10-minute update schedule will be suitable for a frequently used service.

There is also a DOS batch file provided with the same functionality as the Unix script. The batch file is called **StatsCollector.bat**.

Although there are no major security issues, it is good practice to ensure that the script and java program can only be run by a non-privileged account and that others are blocked from using it.

## 4.11 Statistics display

In order for the graphing software to work in the Java Server Page (JSP), `graph.jsp` you will need to have the JBoss JVM run in headless mode. This means that the JVM will not use a graphics sub-system provided by the OS in order to generate the graphics. This only applies to UNIX-like systems. Only a JDK version greater than or equal to 1.4.0 supports this natively. You must add configuration information for `JAVA_OPTS` to `JBOSS_HOME/bin/run.conf` to enable headless mode.

```
JAVA_OPTS="-Djava.awt.headless=true"
```

The additional parameter `'-Djava.awt.headless=true'` can be appended to the existing `JAVA_OPT` declaration in the configuration file.

## 4.12 Linux boot scripts

Boot scripts have been provided for two versions of Linux, providing a Red Hat flavour and a BSD-type flavour for Slackware. These scripts provide the functionality necessary to run JBoss as a non-privileged user and retain the PID for JBoss. The scripts require a modification to the **run.sh** script for JBoss.

### 4.12.1 Pre-requisites

The scripts have been designed to run as the user, **apache**. Create a user **apache** and group **apache** as necessary to ensure the JBoss process and the file access is limited on the Linux system. Alternately you can modify the boot scripts to run as a different user. The scripts should also be modified to reflect the location of the JDK, JBoss and the PID file.

### 4.12.2 Red Hat script

This script is located on the response tester distribution CD under the directory /Scripts and called jboss. Copy it to the /etc/init.d directory of your Red Hat system and use chkconfig to install the JBoss service at the appropriate run level and time in the boot sequence. Typically JBoss should be one of the last services started and one of the first services stopped during boot and shutdown respectively.

### 4.12.3 Slackware script

This script is located on the response tester distribution CD under the directory /scripts and called rc.jboss. Copy it to the /etc/rc.d directory of your Slackware system and insert a call to the script appropriately for rc.K, rc.6 and rc.M. Typically JBoss should be one of the last services started and one of the first services stopped during boot and shutdown respectively.

### 4.12.4 Post installation

After installing your boot scripts, ensure that the permissions are changed for the JBoss directories and files so that the non-privileged user can access the files and appropriately execute the run and shutdown scripts to JBoss.

Modify the boot scripts to reflect the installation directory for JBoss.

```
JBOSS_BIN="/usr/local/JBoss-3.2.2/bin"
```

You will need to modify the run.sh as follows for JBoss 3.2.2 or later:

```
STATUS=10
echo $$ > /tmp/jboss.pid
```

You will need to modify the run.sh as follows for JBoss 3.2.1 or earlier:

```
echo $$ > /tmp/jboss.pid
# Execute the JVM
exec $JAVA $JAVA_OPTS \
```

## 5 Database setup

In order for the response tester software to store results and collate frequency information, it needs a database with some specific tables defined for this purpose. The Amity database services impose a naming syntax to ensure that column and table names do not overlap SQL reserved words. The database may be shared with other applications as long as the tablename does not clash with table names required by the other applications.

Since the system supports many different databases, the exact table creation instructions are not provided. However, most Database Administrators (DBA) will be able to easily perform the necessary translations for the specific database.

The tables described in the following sections need to be created in the same database. The JDBC URL configured in the previous chapter needs to refer to this database.

### 5.1 Response storage

The Amity database services impose a naming syntax to ensure that column and table names do not overlap SQL reserved words.

For the response storage, a table needs to be created called **txResponses**. The columns and column types are given below – the types are Java definitions and you should consult your database documentation to create the relevant database definition.

Column name	Column type
cxIndex	Long
cxTag	String
cxIPAddress	String
cxTime	Long
cxCreated	Timestamp

Finally, a sequence called **sxResponses** must be created. This sequence will be used to populate cxIndex.

### 5.2 Event logging

For the event logging storage, a table needs to be created called **txSEvents**. The columns and column types are given below – the types are Java definitions and you should consult your database documentation to create the relevant database definition.

Column name	Column type
cxIndex	Long
cxSeverity	String
cxSystem	String
cxModule	String
cxMethod	String
cxMessage	String
cxException	String
cxSequel	String
cxVariables	String
cxCreated	Timestamp

Finally, a sequence called **sxSEvents** must be created. This sequence will be used to populate cxIndex.

The event logging service isn't strictly necessary. It is provided as a standard Amity component for detecting problems with the EJB services or the JSP services. This is more critical for delivering debug information in the larger e-Business applications built around Amity FlexCorp.

If the table is not created, the service will write the errors to the standard output. By appropriately configuring your EJB server start scripts, you can capture these errors to a file.

### 5.3 Statistics collection

Should you be running the statistics collection service, you will need to create another table. For the statistics storage, a table needs to be created called **txFrequencies**. The columns and column types are given below – the types are Java definitions and you should consult your database documentation to create the relevant database definition.

Column name	Column type
cxIndex	Long
cxTag	String
cxIPAddress	String
cxLabel	String
cxValue	Long
cxAverage	Double
cxDeviation	Double

A sequence called **sxFrequencies** must also be created. This sequence will be used to populate cxIndex.

### 5.4 Database access

In order to maintain security while allowing the system sufficient privileges to use the database, you should create an account and password that allows the response tester to write and read data from the tables but not necessarily create these tables. The account and password will be the ones used in the connection pool configuration discussed earlier.

## 6 JBoss 3.2.x recommendations

### 6.1 Connection pool

In a medium load environment, we would recommend that the connection pool to begin with a minimum of 5 connections and a maximum of 20 connections. This will allow 5 parallel writes to occur without any new connection having to be established, but will expand up to 20 simultaneous writes if there is sufficient load to warrant it. When the load decreases, the pool will shrink back to 5 connections over time.

This may be tuned as desired to meet normal load expectations for your environment. Although increasing your minimum connection size will reduce write time averages as no new connections need to be established on demand, the trade-off cost is the held resources for open connections to the database.

You achieve this tuning by adding the following tags to the datasource configuration file for your connection pool, **\*-ds.xml**:

```
<min-pool-size>5</min-pool-size>
<max-pool-size>20</max-pool-size>
```

Look at `JBOSS_HOME/docs/examples/jca/generic-ds.xml` for examples on where the tags need to be located in the document structure.

### 6.2 Web application server tuning

Factors that will increase the responsiveness of the response tester reside in the Listener/Connector implementation of the web application server. In particular, the number of available socket connections and the removal of DNS reverse lookup can increase the speed at which HTTP requests can be serviced.

#### 6.2.1 Tomcat

The main area for tuning Tomcat for incoming requests is in **jboss-service.xml** for the Tomcat service. This is located in `JBOSS_HOME/server/default/deploy/jbossweb-tomcat.sar/META-INF`. Due to the number of different sources and changes over releases, the Service ARchive (SAR) for Tomcat could also be named **jbossweb-tomcat41.sar**. The relevant area of the configuration is the 8080 port connector. Usually, the only changes required are to the number of processors available for incoming HTTP requests and to disable lookups. The `minProcessors` and `maxProcessors` corresponds to Apache's `MinSpareServers` and `MaxSpareServers` settings.

```
<!-- A HTTP/1.1 Connector on port 8080 -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  address="${jboss.bind.address}" port="8080" minProcessors="5"
  maxProcessors="100" enableLookups="false" acceptCount="10" debug="0"
  connectionTimeout="20000" useURIVValidationHack="false"/>
```

Refer to the Tomcat website, <http://jakarta.apache.org/tomcat> for more information on configuring the Tomcat connectors and also for more information on configuring Tomcat for HTTPS/SSL, should you require HTTPS. There is also a separate technical note on configuring for Apache-JK integration at <http://www.amityolutions.com.au/document/JK2-technote.pdf>

Note that the connector port binding can also be changed here.

Disable unused connectors by commenting them out. This will reduce the number of threads active in the JBoss system.

You should also disable directory listings for Tomcat by modifying the following in **JBOSS\_HOME/server/default/deploy/jbossweb-tomcat41.sar/web.xml**:

```
<param-name>listings</param-name>
<param-value>>false</param-value>
```

## 6.2.2 Jetty

The main area for tuning Jetty for incoming requests is in **jboss-service.xml** for the Jetty service. This is located in `JBOSS_HOME/server/default/deploy/jbossweb-jetty.sar/META-INF`. Due to the number of different sources and changes over releases, the Service ARchive (SAR) for Jetty could also be named **jboss-jetty.sar**, **jboss-version-jetty-version.sar** or **jetty-plugin.sar**. The relevant area of the configuration is the 8080 port listener. Usually, the only changes required are to the number of threads available for incoming HTTP requests. This corresponds to Apache's `MinSpareServers` and `MaxSpareServers` settings. Do not set `MinThreads` to the same value as `MaxThreads` as `MinThreads` refers to the minimum number of free listeners available.

```
<!-- - - - - - -->
<!-- Add and configure a HTTP listener to port 8080 -->
<!-- The default port can be changed using: java -Djetty.port=80 -->
<!-- - - - - - -->
<Call name="addListener">
  <Arg>
    <New class="org.mortbay.http.SocketListener">
      <Set name="Port"><SystemProperty name="jetty.port" default="8080"/></Set>
      <Set name="MinThreads">5</Set>
      <Set name="MaxThreads">100</Set>
      <Set name="MaxIdleTimeMs">30000</Set>
      <Set name="LowResourcePersistTimeMs">5000</Set>
      <Set name="confidentialPort">8443</Set>
    </New>
  </Arg>
</Call>
```

Refer to the Jetty website, <http://jetty.mortbay.org> for more information on configuring the Jetty listeners and also for more information on configuring Jetty for HTTPS/SSL, should you require HTTPS. There is also a separate technical note on configuring for Apache-JK integration at <http://www.amitysolutions.com.au/document/JK2-technote.pdf>

Note that the listener port binding can also be changed here.

Disable unused listeners by commenting them out. This will reduce the number of threads active in the JBoss system.

You should also disable directory listings for Jetty by modifying the following in **JBOSS\_HOME/server/default/deploy/jbossweb-jetty.sar/defaultweb.xml**

```
<param-name>dirAllowed</param-name>
<param-value>>false</param-value>
```

## 6.2.3 JSP tuning

Both Tomcat and Jetty use the same JSP engine for delivering Java Server Pages. JSP content is compiled at run-time. The compilation occurs the first time the page is called during the lifetime of the application server. However, the compilation time can be lengthy.

There are means to combat this, including the substitution of the Java compiler, javac with the Jikes compiler. In terms of general responsiveness, you can increase the number of available JSP servlets at start-up to meet the request load for JSPs. This setting must be used in conjunction with the number of connectors or listeners configured for the web application server as detailed in the previous two sections. That is; it is of no use to have 10 JSP servlets able to meet JSP requests if the maximum number of Tomcat connectors is only 5.

The bottleneck will be the number of network connections that can be established through available connectors. The default number of JSP servlets is 3 and for heavy loads you may want to increase this to between 5 and 10. However, in most instances the default is adequate.

The configuration file is different on Tomcat and Jetty. In Tomcat, the file is **web.xml** in the SAR. In Jetty the file is **defaultweb.xml** in the SAR. Refer to the previous sections to determine the location of the SAR for your particular web application server.

The snippet relevant to the JSP will look like the following in Tomcat.

```
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
  <init-param>
    <param-name>logVerbosityLevel</param-name>
    <param-value>WARNING</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
</servlet>
```

We would suggest that increasing the time for checking changes to the JSP code, and switching the Jasper 2 system to production rather than development. You can also vary the available JSP servlets at startup.

```
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
  <init-param>
    <param-name>logVerbosityLevel</param-name>
    <param-value>WARNING</param-value>
  </init-param>
  <init-param>
    <param-name>development</param-name>
    <param-value>>false</param-value>
  </init-param>
  <init-param>
    <param-name>checkInterval</param-name>
    <param-value>900</param-value>
  </init-param>
  <load-on-startup>5</load-on-startup>
</servlet>
```

You can obtain further information on Jasper 2 and its configuration from <http://jakarta.apache.org/tomcat/tomcat-4.1-doc/jasper-howto.html>

In order to introduce the Jikes compiler, you will need to obtain Jikes for your OS environment. This would need to be placed in the PATH for your OS so Jikes can be executed by Jasper 2. You can obtain more information about Jikes from <http://oss.software.ibm.com/developerworks/opensource/jikes/>. Jasper 2 requires a Jikes implementation that supports -encoding. This only presents a problem for Windows users. Refer to <http://oss.software.ibm.com/developerworks/opensource/jikes/faq/dev-win32.shtml> for information on building a Jikes Windows version with -encoding support.

The Jikes requirements for Jasper 2 would result in creating the following configuration fragment:

```
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
  <init-param>
    <param-name>logVerbosityLevel</param-name>
    <param-value>WARNING</param-value>
  </init-param>
  <init-param>
    <param-name>development</param-name>
    <param-value>>false</param-value>
  </init-param>
  <init-param>
    <param-name>checkInterval</param-name>
    <param-value>900</param-value>
  </init-param>
  <init-param>
    <param-name>compiler</param-name>
    <param-value>jikes</param-value>
  </init-param>
  <init-param>
    <param-name>javaEncoding</param-name>
    <param-value>iso-8859-1</param-value>
  </init-param>
  <load-on-startup>5</load-on-startup>
</servlet>
```

There are other performance tricks incorporated by Amity Solutions for the response tester. The first trick is to include a special configuration file, **WEB-INF/jetty-web.xml**.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//DTD Configure 1.2//EN"
"http://jetty.mortbay.org/configure_1_2.dtd">
<Configure class="org.jboss.jetty.JBossWebApplicationContext">
  <Call name="getServletHandler">
    <Call name="getServletHolder"><Arg>jsp</Arg>
      <Call name="setInitParameter">
        <Arg>scratchdir</Arg>
        <Arg>/jetty/JSP</Arg>
      </Call>
    </Call>
  </Call>
</Configure>

```

This file is packed in the **responsetester.war** file. It creates the compiled version of the Java Server Pages in /jetty/JSP. Since JBoss does not manage this particular directory, it will not remove the existing pre-compiled files next time JBoss is restarted. Jetty will not recompile the files as they already exist so this will remove the JSP compilation time.

Note that this only works with Jetty. This information is for J2EE deployment engineers in case there is a need to modify the deployment. Since the default build is for J2EE compliance this deployment file is not normally included in the distribution. The second performance trick we use overrides this configuration setting so you need to remove the second configuration we are about to describe for the Jetty specific performance enhancement to work.

The second trick is J2EE compliant and involves the mapping of the JSPs as servlets, defining the JSPs to load on startup. This is the default build for distributions. The settings are defined in the WEB-INF/web.xml of the packed **responsetester.war** file. This information is for J2EE deployment engineers in case there is a need to modify the deployment. However, normally this will not need to be modified by system administrators.

```

<servlet>
  <servlet-name>HOMEJSP</servlet-name>
  <jsp-file>/home.jsp</jsp-file>
  <load-on-startup>3</load-on-startup>
</servlet>
<servlet>
  <servlet-name>TIMEJSP</servlet-name>
  <jsp-file>/time.jsp</jsp-file>
  <load-on-startup>3</load-on-startup>
</servlet>
<servlet>
  <servlet-name>GRAPHJSP</servlet-name>
  <jsp-file>/graph.jsp</jsp-file>
  <load-on-startup>3</load-on-startup>
</servlet>

```

```

<servlet-mapping>
  <servlet-name>HOMEJSP</servlet-name>
  <url-pattern>/home.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>TIMEJSP</servlet-name>
  <url-pattern>/time.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>GRAPHJSP</servlet-name>
  <url-pattern>/graph.jsp</url-pattern>
</servlet-mapping>

```

## 6.3 Security

For all services, it is best practice to ensure that they are not run by a privileged user. Although JBoss developers ensure the servers are secure against attacks, it may be possible that either the JVM or the server may be vulnerable to a buffer overflow or other critical exception that allows an intruder to subvert the server's processes and inherit the privileges of the process owner.

Additionally, where there are configuration files containing sensitive information such as passwords, it is best to ensure that access is limited only to processes that require them.

In particular, ensure that the JDBC datasource configuration files are protected from unauthorised access as the database account and database password are stored in these files.

Running as a non-privileged in UNIX/Linux will prevent

### 6.3.1 JMX-console and Web-console

The JMX-console application provided with JBoss allows monitoring of JBoss Management Bean (MBean) services. This can be used to monitor the health and monitor the state of certain systems such as the datasource for the response tester. It can also be used to stop and start these same systems.

The Web-console allows monitoring of information on available J2EE components. This can be used to determine service bindings for the various response tester components and also the JMX-console and Web-console applications. This application may not exist for early JBoss 3.2.x releases.

By default, the JBoss configuration provides unrestricted access to these web-based consoles. We would advise that authentication be used to protect access to these resources.

JBoss provides several authentication methods. For the expected use of the response tester system, we would recommend the simple file-based user and role mapping. This is suitable when the number of users is small. You can read the JBoss documentation for more details on the other methods for authentication.

**JBOSS\_HOME/server/default/conf/login-config.xml** already has application policies defined for web-console and jmx-console. These define the use of the UsersRolesLoginModule that relates to the file-based user and role mapping. We provide the snippet here for reference but you shouldn't be required to make any changes unless you need hashed passwords.

```

<!-- A template configuration for the jmx-console web application. This
     defaults to the UsersRolesLoginModule the same as other and should be
     changed to a stronger authentication mechanism as required.
-->
<application-policy name = "jmx-console">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
      flag = "required" />
  </authentication>
</application-policy>
<!-- A template configuration for the web-console web application. This
     defaults to the UsersRolesLoginModule the same as other and should be
     changed to a stronger authentication mechanism as required.
-->
<application-policy name = "web-console">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
      flag = "required" />
  </authentication>
</application-policy>

```

For hash encoding you will need to insert an element for the authentication.

```
<module-option name="hashEncoding">base64</module-option>
```

Refer to the JBoss documentation for more information on the encoding. Currently, only base64 and hex encoding are supported.

You need to create two files that contain the authentication and mapping information. The user information is contained in **JBOSS\_HOME/server/default/conf/users.properties**. This contains the username and the password, with one line per user. The following example assumes no hash encoding is employed.

```

jonb=mulberry
michaeli=tangerine

```

The role information is contained in **JBOSS\_HOME/server/default/conf/roles.properties**. This contains the username and the roles that user is granted when authenticated. The example shows the relationship between users and roles in the file.

```

jonb=JBossAdmin,AmityAdmin
michaeli=JBossAdmin

```

Create the user information as necessary for your requirements. The users must be given the role, **JBossAdmin** as this is the pre-defined role for the two console applications.

In order to enable the security and authentication for the JMX-console application, you will need to edit the two deployment descriptors for the web application. Uncomment the security-domain element in **JBOSS\_HOME/server/default/deploy/jmx-console.war/WEB-INF/jboss-web.xml**.

```

<jboss-web>
  <!-- Uncomment the security-domain to enable security. You will
  need to edit the htmladaptor login configuration to setup the
  login modules used to authentication users. -->
  <security-domain>java:/jaas/jmx-console</security-domain>
</jboss-web>

```

Note that this links to the definition in **login-config.xml**.

Uncomment the security-constraint element in **JBOSS\_HOME/server/default/deploy/jmx-console.war/WEB-INF/web.xml**. This links to the security role assigned to the authenticated user, and restricts access to the application to authenticated users.

```

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

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>JBoss JMX Console</realm-name>
</login-config>

<security-role>
  <role-name>JBossAdmin</role-name>
</security-role>

```

For the web console application, you will have to perform the same task. However, the web console application has been delivered in a packed format so you will need to unpack it first to make the changes.

Create a local directory to hold the unpacked application. Unpack the file contained in **JBOSS\_HOME/server/default/deploy/management/web-console.war** into this directory, using the Java jar tool. You can edit the **WEB-INF/web.xml** and **WEB-INF/jboss-web.xml** files as previously done with the JMX-console application. The following gives a UNIX/Linux example.

```
cd ~
mkdir web-console
cd web-console
jar xvf JBOSS_HOME/server/default/deploy/management/web-console.war
```

After making changes to the deployment descriptors, package up the files and copy the deployment package back to the JBoss run-time location.

```
jar cvf web-console.war *
cp web-console.war JBOSS_HOME/server/default/deploy/management
```

Test the applications by accessing them locally using a web browser.

<http://localhost:8080/jmx-console>

<http://localhost:8080/web-console>

Both applications should challenge you for a username and password. Try using one of your created users to test that authentication works. Finally, ensure that the access to users.properties and roles.properties are restricted.

This completes the work required to secure the console applications for JBoss.

## 6.4 Unused services

The default JBoss run-time configuration still has many services that may remain unused. As recommended, you should disable any of the web application server connectors or listeners. Refer to section 6.2 for more information.

The response tester does not use the scheduler service. Should you be using JBoss only for the response tester application, you may remove the following files to disable these services.

From JBOSS\_HOME/server/default/lib:

- scheduler-plugin-example.jar
- scheduler-plugin.jar

From JBOSS\_HOME/server/default/deploy:

- schedule-manager-service.xml
- scheduler-service.xml

For the removal of other services, refer to the JBoss documentation or contact Amity Solutions for more information.

## 6.5 Logging

JBoss extensively uses the Apache Log4J libraries for controlling logging of information. Refer to <http://jakarta.apache.org/log4j/docs/index.html> for more information on Log4J and configuration details. Edit the **JBOSS\_HOME/server/default/conf/log4j.xml** file to produce the appropriate logging levels. There are many examples for configurations in the file.

We recommend that an INFO level configuration is used for logging information to **JBOSS\_HOME/server/default/log/server.log**. The default configuration is for a rolling log, with a new log file created every day, and the old one renamed with an appended date, so some administration is required to manage the growing number of logs. This is defined by declaring a rolling file appender. An example configuration with our recommendations has been provided.

```

<!-- A time/date based rolling appender -->
<appender name="FILE" class="org.jboss.logging.appender.DailyRollingFileAppender">
  <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
  <param name="File" value="${jboss.server.home.dir}/log/server.log"/>
  <param name="Append" value="false"/>
  <param name="Threshold" value="INFO"/>

  <!-- Rollover at midnight each day -->
  <param name="DatePattern" value="'.'yyyy-MM-dd"/>

  <!-- Rollover at the top of each hour -->
  <param name="DatePattern" value="'.'yyyy-MM-dd-HH"/>
  -->

  <layout class="org.apache.log4j.PatternLayout">
    <!-- The default pattern: Date Priority [Category] Message\n -->
    <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>

    <!-- The full pattern: Date MS Priority [Category] (Thread:NDC) Message\n -->
    <param name="ConversionPattern" value="%d %-5r %-5p [%c] (%t:%x) %m%n"/>
    -->
  </layout>
</appender>

```

The examples also show how to log errors to an e-mail destination and to the SNMP trap handler provided in JBoss 3.2.2. You also have fine-grain control over events logged through the use of categories. These are also shown by example in the configuration file.

## 7 Standalone Tomcat configuration

### 7.1 Tomcat 4.1

Install Tomcat 4.1 as per the Tomcat installation instructions. These are supplied with the Tomcat package and may be obtained from the Apache website, <http://jakarta.apache.org/tomcat>. As a standard installation, with minimal additional configuration, install Tomcat on the server platform. As well as being a servlet/JSP application server, it operates as a full webserver so you will not need to install a separate webserver and configure that to operate with Tomcat.

For a UNIX/Linux installation, we would recommend either unpacking the contents of the package into `/opt/Tomcat-version` or `/usr/local/Tomcat-version`, using the appropriate version number for your Tomcat installation. The latest version is Tomcat 4.1.29 so the installation directory would be `/usr/local/Tomcat-4.1.29`.

Similarly, you should install JBoss in an appropriate directory on your Microsoft server. You should not install your application server in a directory path that contains a space.

From now on, we will refer to the Tomcat root directory as `CATALINA_HOME`, for historical reasons. Taking our previous example, this would mean that `CATALINA_HOME=/usr/local/Tomcat-4.1.29`.

You do not need to use a standalone copy of Tomcat as JBoss provides an embedded web application server. However, there may be situations where a separated environment is required.

In a Microsoft environment do not install Tomcat in a directory path that contains a space. This will usually cause Java Naming and Directory Interface (JNDI) lookup problems for Tomcat when trying to locate components on the JBoss application server. The usual error associated with this is `java.net.MalformedURLException: no protocol`, followed by a file path that has been truncated due to the space in the path.

#### 7.1.1 Amity support components

The following Amity support components must be installed in the Tomcat `CATALINA_HOME/shared/lib` directory located off the main Tomcat directory:

- `eventloggerclient.jar`
- `responsesclient.jar`
- `frequenciesclient.jar`
- `tomcatejbref.jar`
- `flexweb.jar`
- `access.sqlmanager.jar` (if running Access or SQL Server)
- `oracle.sqlmanager.jar` (if running Oracle)
- `db2.sqlmanager.jar` (if running DB2 UDB 7.2)
- `postgresql.sqlmanager.jar` (if running PostgreSQL)

Later versions of Tomcat may not have the `CATALINA_HOME/shared/lib` directory. In this case, copy the libraries into `CATALINA_HOME/common/lib`.

#### 7.1.2 Support components

In order for the servlets and Java Server Pages (JSP) to be able to communicate with the EJBs located in the JBoss server, the main JBoss client library must also be installed in the Tomcat `CATALINA_HOME/shared/lib` directory. This file is included in the JBoss distribution and is called `JBOSS_HOME/client/jbossall-client.jar`. The charting library, `JOpenChart.jar` must also be copied into the same directory.

Later versions of Tomcat may not have the **CATALINA\_HOME/shared/lib** directory. In this case, copy the libraries into **CATALINA\_HOME/common/lib**.

### 7.1.3 Deployment modifications

The response tester web application has been updated to take advantage of an integrated runtime environment containing both the web application server and the EJB server. So there are some modifications required to allow the response tester application to operate in a distributed environment. The nature of the design allows customization without the intervention of personnel from the vendor.

There is one area for changes. The changes relate to the coding that accesses the remote components located on the JBoss server. The two JSPs for the **responsetester.war** file must be modified. In order to modify these, you must unpack **responsetester.war** into a working directory first. Use the Java jar utility to do this. The following example demonstrates the steps in a UNIX/Linux environment.

```
mkdir temp
cd temp
jar xvf ../responsetester.war
```

There are two files to modify, **time.jsp** and **graph.jsp**. The changes redirect the JNDI lookups to the remote JBoss server and make use of the remote interfaces for the components. Substitute **jnp://myhost:port** with the hostname or address of the server on which JBoss resides, and the port on which the JNP naming service (JNDI) is available. The default is port 1099. Refer to section 4.6 for more information. We also replace the local JNDI names with the JBoss global JNDI names. The global JNDI names and components to which they belong can be verified by using the JBoss JMX-console or the JBoss Web-console applications.

For **time.jsp** -

Replace:

```
com.amity.flexcorp.flexweb.ReferenceFactory,
com.amity.responsetester.responses.Response,
com.amity.responsetester.responses.ResponsesLocal,
com.amity.responsetester.responses.ResponsesLocalHome"
```

With:

```
javax.rmi.PortableRemoteObject,
com.amity.flexcorp.flexweb.ReferenceFactory,
com.amity.responsetester.responses.Response,
com.amity.responsetester.responses.Responses,
com.amity.responsetester.responses.ResponsesHome"
```

Replace:

```
ResponsesLocalHome home = null;
public void jspInit()
{
    // Get handle to remote object
    ReferenceFactory factory = new ReferenceFactory();
    home = (ResponsesLocalHome)factory.getReference
        ("java:comp/env/ejb/amity/responsetester/Responses");
}
```

With:

```

ResponsesHome home = null;
public void jspInit()
{
    // Get handle to remote object
    ReferenceFactory factory = new ReferenceFactory
        ("org.jnp.interfaces.NamingContextFactory", "jnp://myhost:port",
        "org.jboss.naming:org.jnp.interfaces", false);
    try
    {
        Object reference = factory.getReference
            ("ejb/amity/responsetester/remote/Responses");
        home = (ResponsesHome)PortableRemoteObject.narrow
            (reference, ResponsesHome.class);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

Replace:

```
ResponsesLocal responseManager = home.create();
```

With:

```
Responses responseManager = home.create();
```

For **graph.jsp** -

Replace:

```

com.amity.flexcorp.flexweb.ReferenceFactory,
com.amity.responsetester.responses.Frequency,
com.amity.responsetester.responses.FrequenciesLocal,
com.amity.responsetester.responses.FrequenciesLocalHome"

```

With:

```

javax.rmi.PortableRemoteObject,
com.amity.flexcorp.flexweb.ReferenceFactory,
com.amity.responsetester.responses.Frequency,
com.amity.responsetester.responses.Frequencies,
com.amity.responsetester.responses.FrequenciesHome"

```

**Replace:**

```
FrequenciesLocalHome home = null;
public void jspInit()
{
    // Get handle to remote object
    ReferenceFactory factory = new ReferenceFactory();
    home = (FrequenciesLocalHome)factory.getReference
        ("java:comp/env/ejb/amity/responsetester/Frequencies");
}

```

**With:**

```
FrequenciesHome home = null;
public void jspInit()
{
    // Get handle to remote object
    ReferenceFactory factory = new ReferenceFactory
        ("org.jnp.interfaces.NamingContextFactory", "jnp://myhost:port",
        "org.jboss.naming:org.jnp.interfaces", false);
    try
    {
        Object reference = factory.getReference
            ("ejb/amity/responsetester/remote/Frequencies");
        home = (FrequenciesHome)PortableRemoteObject.narrow
            (reference, FrequenciesHome.class);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

**Replace:**

```
FrequenciesLocal frequencyManager = home.create();
```

**With:**

```
Frequencies frequencyManager = home.create();
```

Once these changes are made, pack the files and deploy the modified **responsetester.war** to CATALINA\_HOME/webapps.

```
jar cvf responsetester.war WEB-INF css images js *.jsp *.html
```

## 7.1.4 Response tester package deployment

A WAR file contains the deployment package for the application. Unlike JBoss, Tomcat does not have a hot deployment service. You will either need to stop Tomcat and place the WAR file, **responsetester.war** in the `CATALINA_HOME/webapps` directory or use the Tomcat manager console that is available in later versions of Tomcat. Start Tomcat by executing the 'start' script – either **startup.bat** in a Microsoft environment or **startup.sh** in a Unix environment. The script is located in the **bin** directory of the Tomcat installation.

A successful start of Tomcat will cause the **responsetester.war** package to be unpacked into a directory called **responsetester**. Consult the Tomcat documentation for more information on the deployment of web applications in Tomcat.

You can view the status of the response tester web application using the Tomcat administration console or the Tomcat manager console available in later releases of Tomcat. Access to these consoles is disabled by default. In order to enable them, you will need to modify the **CATALINA\_HOME/conf/tomcat-users.xml** file. This will involve adding some roles and authorised users. An example is given below that provides a single user with access to the manager and administration consoles:

```
<role rolename="manager" />
<role rolename="admin" />
<user username="jonb" password="mulberry" roles="manager,admin" />
```

## 7.1.5 Special considerations

We would recommend that you use Tomcat 4.1.12 or higher and configure Tomcat to use the Jikes Java compiler for JSP compilation. The Tomcat distribution and website have instructions on how to accomplish this. The standard JDK supplied compiler causes a memory leak when invoked by the Tomcat JSP compilation engine. The Jikes compiler does not suffer from this problem and this will increase the capability of an already robust application server. The memory problem is only of significance when there are hundreds of JSPs but we feel that taking this care will ensure that no problems can be attributed to your Tomcat installation. Jikes is also a faster compiler than the standard JDK-supplied compiler.

You should also configure Tomcat for a production install so that application initialization times are minimised. Refer to the Tomcat configuration documentation for more information on this. The production configuration ensures that the JSPs are not recompiled every time the server is rebooted.

As an alternative to using Jikes, upgrade your Sun Java 2 installation to Sun Java 2 JDK 1.4.1. The Sun documentation indicates that this release addresses the memory leakage problem. It also deals with instances of a particular process halt in the Tomcat server.

In environments supported by the IBM Java 2 SDK 1.4.x, we would recommend using this as it is a fast and robust implementation.

In order for the charting servlet to work under Unix, you will need to enable the Tomcat JVM to run in a headless environment; that is, without a graphics environment such as X-Windows as is required normally by Java graphics routines. You do need JDK 1.4.0 or greater for this to work. You will need to add to the Tomcat environment variable, `CATALINA_OPTS` containing the following:

```
CATALINA_OPTS = "${CATALINA_OPTS} -Djava.awt.headless=true"
```

This can be configured in **catalina.sh** or **catalina.bat**, depending on your environment.

Tomcat uses a number of ports for providing the JK, HTTP and HTTPS services. It also has a control port for shutting down the Tomcat server.

- 8005 Shutdown port
- 8009 Embedded servlet container JK connector
- 8080 Embedded servlet container connector (default HTTP service)
- 8443 Embedded servlet container SSL connector (default HTTPS service)

The port bindings and services available can be configured in **CATALINA\_HOME/conf/server.xml**. We suggest that you refer to the Tomcat documentation at <http://jakarta.apache.org/tomcat> for further information on configuration of the Tomcat server.

## 7.1.6 Alternative modifications

An alternative to the previously detailed deployment modifications reduces the number of programming changes, although it introduces deployment configuration changes. This may be more suitable as it allows greater flexibility to swap the same deployment package **responsetester.war** between an integrated JBoss-Tomcat or JBoss-Jetty installation and a standalone Tomcat installation. There are three areas for change.

The first area of changes relate to the coding that accesses the remote components located on the JBoss server. The two JSPs for the **responsetester.war** file must be modified. In order to modify these, you must unpack **responsetester.war** into a working directory first. Use the Java jar utility to do this. The following example demonstrates the steps in a UNIX/Linux environment.

```
mkdir temp
cd temp
jar xvf ../responsetester.war
```

There are two files to modify, **time.jsp** and **graph.jsp**. The changes redirect the JNDI lookups to the remote JBoss server and make use of the remote interfaces for the components. Substitute **jnp://myhost:port** with the hostname or address of the server on which JBoss resides, and the port on which the JNP naming service (JNDI) is available. The default is port 1099. Refer to section 4.6 for more information. We also replace the local JNDI names with the JBoss global JNDI names. The global JNDI names and components to which they belong can be verified by using the JBoss JMX-console or the JBoss Web-console applications.

For **time.jsp** -

Replace:

```
com.amity.flexcorp.flexweb.ReferenceFactory,
com.amity.responsetester.responses.Response,
com.amity.responsetester.responses.ResponsesLocal,
com.amity.responsetester.responses.ResponsesLocalHome"
```

With:

```
javax.naming.InitialContext,
javax.rmi.PortableRemoteObject,
com.amity.flexcorp.flexweb.ReferenceFactory,
com.amity.responsetester.responses.Response,
com.amity.responsetester.responses.Responses,
com.amity.responsetester.responses.ResponsesHome"
```

**Replace:**

```

ResponsesLocalHome home = null;
public void jspInit()
{
    // Get handle to remote object
    ReferenceFactory factory = new ReferenceFactory();
    home = (ResponsesLocalHome)factory.getReference
        ("java:comp/env/ejb/amity/responsetester/Responses");
}

```

**With:**

```

ResponsesHome home = null;
public void jspInit()
{
    // Get handle to remote object
    try
    {
        Object reference = (new InitialContext()).lookup
            ("java:comp/env/ejb/amity/responsetester/Responses");
        home = (ResponsesHome)PortableRemoteObject.narrow
            (reference, ResponsesHome.class);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

**Replace:**

```
ResponsesLocal responseManager = home.create();
```

**With:**

```
Responses responseManager = home.create();
```

**For `graph.jsp` -****Replace:**

```

com.amity.flexcorp.flexweb.ReferenceFactory,
com.amity.responsetester.responses.Frequency,
com.amity.responsetester.responses.FrequenciesLocal,
com.amity.responsetester.responses.FrequenciesLocalHome"

```

**With:**

```

    javax.naming.InitialContext,
    javax.rmi.PortableRemoteObject,
    com.amity.flexcorp.flexweb.ReferenceFactory,
    com.amity.responsetester.responses.Frequency,
    com.amity.responsetester.responses.Frequencies,
    com.amity.responsetester.responses.FrequenciesHome"

```

**Replace:**

```

FrequenciesLocalHome home = null;
public void jspInit()
{
    // Get handle to remote object
    ReferenceFactory factory = new ReferenceFactory();
    home = (FrequenciesLocalHome)factory.getReference
        ("java:comp/env/ejb/amity/responsetester/Frequencies");
}

```

**With:**

```

FrequenciesHome home = null;
public void jspInit()
{
    // Get handle to remote object
    try
    {
        Object reference = (new InitialContext()).lookup
            ("java:comp/env/ejb/amity/responsetester/Frequencies");
        home = (FrequenciesHome)PortableRemoteObject.narrow
            (reference, FrequenciesHome.class);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

**Replace:**

```

FrequenciesLocal frequencyManager = home.create();

```

**With:**

```

Frequencies frequencyManager = home.create();

```

The second area for change is the deployment descriptors for the response tester web application. These need to be modified to reflect the use of the remote interfaces for the EJBs rather than the local interfaces.

For **WEB-INF/web.xml** -

## Replace:

```

<ejb-local-ref>
  <ejb-ref-name>ejb/amity/responsetester/Responses</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.amity.responsetester.responses.ResponsesLocalHome</local-home>
  <local>com.amity.responsetester.responses.ResponsesLocal</local>
</ejb-local-ref>
<ejb-local-ref>
  <ejb-ref-name>ejb/amity/responsetester/Frequencies</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.amity.responsetester.frequencies.FrequenciesLocalHome</local-home>
  <local>com.amity.responsetester.frequencies.FrequenciesLocal</local>
</ejb-local-ref>

```

## With:

```

<ejb-ref>
  <ejb-ref-name>ejb/amity/responsetester/Responses</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.amity.responsetester.responses.ResponsesLocalHome</home>
  <remote>com.amity.responsetester.responses.ResponsesLocal</remote>
</ejb-ref>
<ejb-ref>
  <ejb-ref-name>ejb/amity/responsetester/Frequencies</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.amity.responsetester.frequencies.FrequenciesHome</home>
  <remote>com.amity.responsetester.frequencies.Frequencies</remote>
</ejb-ref>

```

For **WEB-INF/jboss-web.xml** -

## Replace:

```

<ejb-local-ref>
  <ejb-ref-name>ejb/amity/responsetester/Responses</ejb-ref-name>
  <local-jndi-name>ejb/amity/responsetester/local/Responses</local-jndi-name>
</ejb-local-ref>
<ejb-local-ref>
  <ejb-ref-name>ejb/amity/responsetester/Frequencies</ejb-ref-name>
  <local-jndi-name>ejb/amity/responsetester/local/Frequencies</local-jndi-name>
</ejb-local-ref>

```

With:

```
<ejb-ref>
  <ejb-ref-name>ejb/amity/responsetester/Responses</ejb-ref-name>
  <jndi-name>ejb/amity/responsetester/remote/Responses</jndi-name>
</ejb-ref>
<ejb-ref>
  <ejb-ref-name>ejb/amity/responsetester/Frequencies</ejb-ref-name>
  <jndi-name>ejb/amity/responsetester/remote/Frequencies</jndi-name>
</ejb-ref>
```

Once these changes are made, pack the files and deploy the modified **responsetester.war** to `CATALINA_HOME/webapps`.

```
jar cvf responsetester.war WEB-INF css images js *.jsp *.html
```

The final area for change is to add the JBoss JNDI lookup capability to Tomcat. This involves adding a plugin library and modifying **CATALINA\_HOME/conf/server.xml**.

Copy **tomcatejbref.jar** from the distribution CD to `CATALINA_HOME/shared/lib` or to `CATALINA_HOME/common/lib` as appropriate for your Tomcat server. Refer to section 7.1.1 for more details on the correct library directory for your Tomcat installation.

Add the following context to `server.xml`.

```
<Context path="/responsetester" docBase="responsetester" debug="0"
  reloadable="true" crossContext="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
    prefix="localhost_responsetester_log." suffix=".txt"
    timestamp="true"/>
  <Ejb name="ejb/amity/responsetester/Responses" type="Session"
    home="com.amity.responsetester.responses.ResponsesHome"
    remote="com.amity.responsetester.responses.Responses"/>
  <ResourceParams name="ejb/amity/responsetester/Responses">
    <parameter>
      <name>factory</name>
      <value>com.amity.utils.tomcat.EJBRefFactory</value>
    </parameter>
    <parameter>
      <name>java.naming.provider.url</name>
      <value>jnp://myserver:myport</value>
    </parameter>
    <parameter>
      <name>jndi-name</name>
      <value>ejb/amity/responsetester/remote/Responses</value>
    </parameter>
  </ResourceParams>
```

```

<Ejb name="ejb/amity/responsetester/Frequencies" type="Session"
    home="com.amity.responsetester.frequencies.FrequenciesHome"
    remote="com.amity.responsetester.frequencies.Frequencies"/>
<ResourceParams name="ejb/amity/responsetester/Frequencies">
    <parameter>
        <name>factory</name>
        <value>com.amity.utils.tomcat.EJBRefFactory</value>
    </parameter>
    <parameter>
        <name>java.naming.provider.url</name>
        <value>jnp://myserver:myport</value>
    </parameter>
    <parameter>
        <name>jndi-name</name>
        <value>ejb/amity/responsetester/remote/Frequencies</value>
    </parameter>
</ResourceParams>
</Context>

```

Replace **myserver** with the IP address or name of the server that has JBoss installed on it. Replace **myport** with the port to which the JNP naming server is bound – the default is port 1099. Refer to section 4.6 for details on the ports and services.

Restart Tomcat once these changes have been made.

Refer to <http://www.amityolutions.com.au/documents/JBossTomcatJNDI.pdf> for more information on using the JBoss JNDI service from a standalone Tomcat server. Also refer to the Tomcat documentation for more information on JNDI resources and context configuration at <http://jakarta.apache.org/tomcat>.

Please note that the Tomcat configuration requires more maintenance and that any new contexts require specific definition so that the resources are correctly resolved.

## 8 Troubleshooting for JBoss

### 8.1 JBoss/Tomcat configuration checklist

Parameter	Default Value	Custom
JBoss		
RMI naming service	1098	
Naming Service	1099	
Hypersonic JDBC service	N/A (1701)	
RMI object service	4444	
Pooled invoker	4445	
Web Service	8083	
JMS OIL service	8090	
JMS UIL service	N/A (8091)	
JMS OIL2 service	8092	
JMS UIL2 service	8093	
IP address	jboss.bind.address	
Tomcat/Jetty		
Standard web port	8080	
Standard web redirect port	8443	
Standard SSL web port	8443	
Standard SSL web redirect port	-	
AJP connector port	8009	
AJP connector redirect port	8443	
IP address	jboss.bind.address	

### 8.2 Specific installation cases

Other than issues with standard operation of the web application servers, the only area of reconfiguration in the application will be within the configurable JSPs.

- If you cannot access the web pages for the application
  - Is Tomcat running? Tomcat may not have started successfully or may have been shut down. Please check with your system administrator for correct configuration of Tomcat and check the Tomcat logs for any issues with the start up.
  - Have you checked the web port setting for Tomcat? Many Unix servers will not allow you to bind the Tomcat service to ports below 1024 for security reasons when running as a non-privileged user. Also, you may have other security restrictions in place that prevents you from doing this. Please check with your system administrator and network administrator for any security issues.
- If you cannot write or read data from the database
  - Have you changed the port that the JBoss naming service is operating from (1099) or is the Tomcat server operating on a different machine to the JBoss server?

If either of these cases have occurred you will need to modify the **time.jsp** or your custom JSPs to connect to the JBoss server. You must replace the line,

```
ReferenceFactory factory = new ReferenceFactory();
```

With the line:

```
ReferenceFactory factory = new ReferenceFactory(  
    "org.jnp.interfaces.NamingContextFactory",  
    "jnp://myaddress:myport",  
    "org.jboss.naming:org.jnp.interfaces");
```

Replace myaddress and myport with the appropriate address of the JBoss server and the JBoss server port for the naming service.

Note that this problem is usually only encountered with the standalone Tomcat configuration.

- Have you any security issues locking the naming service port for JBoss on your server? You may need to check with your system administrator to confirm that you can configure JBoss to use that port. You can also test connectivity by telneting to the JBoss port from your Tomcat server.
  - Have you configured the database for the tables you are attempting to use? Check the JBoss logs for any indication that the database tables do not exist and rectify the problem accordingly.
  - Are there problems connecting to the database? Check the JBoss logs for any indication that the database is not contactable and rectify the connection problem. This problem may be a result of an incorrect JDBC URL, an incorrect version of the JDBC driver or there may be a network security related problem where the JBoss server is blocked from connecting to the database. Please confirm all these issues with your database administrator and your network administrator.
- If you cannot get the chart software to operate
  - Ensure that you have taken adequate steps to allow the software to operate in a headless environment. Refer to the recommendations in section 4.11. If you are using a JVM that is older than 1.4.0, you will need to find the documentation for your JVM that describes running Java software in a headless environment.

## 8.3 Operational troubleshooting

These troubleshooting tips assume that your service was operational and you have encountered a problem during the operational lifetime of the system.

### 8.3.1 Unable to write to the database

The response tester provides the following message:

**Unable to write the record to the Response Test database.**

The most likely causes for these issues are:

- The database is no longer operational
- The database access name and password for JBoss has changed
- JBoss has lost the connection pool to the database, either through a network outage or through the database server having been restarted, stopped or moved

Check the access details for the database and compare this with the information you compiled for the connection pooling in section 4.2.2. Consult with your database administrator. When confirmed, restart JBoss to re-establish the database connection pool.

### 8.3.2 Unable to find the writer service

The response tester provides the following message:

**Unable to connect to the Response Test transaction writer service.**

This situation is more likely to occur in a separated JBoss/Tomcat or JBoss/Jetty configuration. The most likely causes for the issue are:

- JBoss has been restarted and the web application server has not been reset
- JBoss has been stopped

Check that JBoss is still operational, and if not, restart it. Check that the response tester components are installed and running. Restart Jetty/Tomcat.

### 8.3.3 JSP pages don't display after some time

The symptom is that the pages display without problem after a reboot of JBoss and then they stop displaying, at some point in time. This usually occurs with a Jetty installation when the UNIX/Linux server clears out the /tmp directory, removing the servlet container's compiled code for the pages.

Check that Jetty or Tomcat is still running. Should they not be running, then the issue is because the web application server is no longer available.

Should you suspect that the problem is with the compiled code being removed from the temporary directory, you can force Jetty to create the temp files elsewhere using by defining the Java environment property, `java.io.tmpdir`. You can do this by appending to the `JAVA_OPTS` definition in **JBOSS\_HOME/bin/run.conf**. The following shows a setting for headless operation and for the temp directory for JVM files to be located at /jetty.

```
JAVA_OPTS="-Djava.awt.headless=true -Djava.io.tmpdir=/jetty"
```

You will need to create the /jetty directory with permissions such that the JBoss process can write into the directory.

### 8.3.4 JBoss performs a clean shutdown by itself

The symptom is that the JBoss server shuts down after some time. Checking the logs shows that a clean shutdown sequence was initiated but the shutdown script was not run and no user initiated any manual shutdown by sending the process a KILL signal.

This is more likely to occur in a UNIX/Linux environment and is caused by the OS generating a signal that the JVM interprets as a KILL signal. The JBoss microkernel has been designed to perform an orderly shutdown based on such a signal. However, sometimes due to the OS and JVM interoperation, a spurious signal may be misinterpreted. The JVM operation can be made less sensitive by configuring it to be less tightly coupled to the OS signals. You can achieve this by adding a switch to the `JAVA_OPTS` definition in **JBOSS\_HOME/bin/run.conf**. The following shows a setting for headless operation, for the temp directory for JVM files to be located at /jetty and to reduce the use of OS signals.

```
JAVA_OPTS="-Djava.awt.headless=true -Djava.io.tmpdir=/jetty -Xrs"
```

You can investigate further by reading the IBM developer solutions paper on signal handling at <http://www-106.ibm.com/developerworks/ibm/library/i-signalhandling/>. You can also examine other Java -X options by running the following:

```
java -X
```