Leading Open Source Middleware

JOnAS 5 Configuration guide

JOnAS Team ( )

- September 2009 -

Table of Contents

List of Examples

Chapter 1. Introduction

1.1. Configuring JOnAS

Configuration is a task that may be more or less complex. Configuring a unique instance is obviously easier than configuring a cluster of servers.

Configuration task consists mainly in customizing a set of JOnAS configuration files that compose the JOnAS environment see Section 2.1, “Configuring JOnAS Environment”.

First of all, some terms used is this document must be defined:

1.2. Terminology

1.2.1. Server or JOnAS instance

A server, or JOnAS instance, is a java process started via the *jonas start* command, or via the administration tool Java EE.

Several servers may run on the same physical host.

1.2.2. Service

When a server starts, services are started.

A service typically provides system resources to containers. Most of the components of the JOnAS application server are pre-defined services. However, it is possible and easy for an advanced user to define a new service and to integrate it into JOnAS.

JOnAS services are manageable through JMX.

1.2.3. Container

A container consists of a set of Java classes that implement the Java EE specification. The role of the container is to provide the facilities for executing Java EE components.

There are three types of containers:

- EJB container in which Enterprise JavaBeans are deployed and run
- Web container for JSPs and servlets
- Client container

1.2.4. Domain

A domain represents an administration perimeter which is under the control of an administration authority.

This perimeter contains management targets like servers and clusters.

If a domain contains several elements, it provides at least one common administration point represented by a master server.

1.2.5. Master server

A master is a JOnAS instance having particular management capabilities within the domain:

- it is aware of the domain's topology
- it allows management and monitoring of all the elements belonging to the domain

1.2.6. Cluster

A cluster is a group of JOnAS servers having common properties within a domain. It usually allows to run a J2EE application, or a J2EE module, on the cluster members as if they were a single server. The objective is to achieve applications scalability and high availability.

JOnAS supports several cluster types:

- Clusters for Web level load-balancing
- Clusters for high availability of Web components
- Clusters for EJB level load-balancing
- Clusters for high availability of EJB components
- Clusters for JMS destination scalability and high availability
- Clusters for administration purpose which facilitate management operations like deployment / undeployment.

From the administrator point of view, a cluster represents a single administration target.

Note that a particular JOnAS server may belong to zero, one or more clusters.

Chapter 2. Configuring a JOnAS instance

JOnAS is pre-configured and ready to be used directly. The Getting Started book [GettingStarted.html#GettingStarted] has shown that a very sample example may be run after JOnAS installation without any configuration task. But as soon as your application needs to use resources specific to the execution environment, configuration is mandatory.

In this chapter we will see in a first part where are the configuration files and then what that can be configured

2.1. Configuring JOnAS Environment

JOnAS distribution contains a number of configuration files in `$JONAS_ROOT/conf` directory. These files can be edited to change the default configuration. However, it is recommended that the configuration files needed by a specific application running on JOnAS be placed in a separate location. This is done by using an additional environment variable called JONAS_BASE.

JOnAS configuration files are read from the $JONAS_BASE/conf directory. If JONAS_BASE is not defined, it is automatically initialized to $JONAS_ROOT.

2.1.1. JONAS_ROOT structure

The installation directory (JONAS_ROOT) has the following structure:

- the `deploy/` directory

  The main location used for deployment.

  At JOnAS startup, all deployment plans, Java EE archives and OSGi bundles are deployed in the following order:

  1. Deployment plan repositories
  2. OSGi bundles
  3. RAR archives
  4. Deployment plan resources
  5. EJB archives
  6. WAR archives
  7. EAR archives

  Note

  For each category, file names are chosen in alphabetical order

  This directory is periodically polled in order to deploy new archives. For more information have a look at the depmonitor service configuration [configuration_guide.html#services.depmonitor.config]

- the `bin/` directory

  contains the scripts used to launch JOnAS (Unix and Windows scripts).

- the `conf/` directory

  contains the JOnAS configuration files.

- the `examples/` directory

  this sub tree containing all the JOnAS examples that are described in ???

- the `lib/` directory [1]

  Used for extending class loaders. It contains five sub directories:

Directory	Description
`bootstrap/`	Jars loaded by the JOnAS bootstrap
`common/`	Legacy directory where Ant tasks are stored
`endorsed/`	Jars overridding JVM libraries
`ext/`	For non-bundle extensions
`internal-ee-tld/`	Internal use only !

- the `logs/` directory

  where the log files are created at run-time (when the JONAS_ROOT is used as a JONAS_BASE)

- the `templates/` directory

  this sub tree contains the following subdirectories used by JOnAS during the generation process (eg, JONAS_BASE generation).

  - `conf/` is an empty template of the JONAS_BASE structure used by tools able to create a JONAS_BASE environment.
  - `newjb/` contains the configuration files for creating a JONAS_BASE environment.
  - `newjc/` contains the configuration files for creating a cluster environment.

- the `repositories/` directory

  this sub tree contains the following repositories used to store OSGi bundles, Java EE applications and deployment plans.

  - `maven2-internal/` contains both OSGi bundles and applications (jonasAdmin, documentation, ...) for JOnAS. It is used for internal purpose and should not be modified. This directory is structured as a Maven2 repository.
  - `url-internal/` contains the deployment plans of each JOnAS services. It is used for internal purpose and should not be modified.
  - `<repository-id>/` contains archives downloaded through deployment plans from this repository.

2.1.2. JONAS_BASE structure

JONAS_BASE has the following structure:

- the `conf/` directory

  contains JOnAS configuration files.

- the `deploy/` directory

is the main location used for deployment.

At JOnAS startup time all the deployment plans, Java EE archives and OSGi bundles are deployed in the following order:

1. Deployment plan repositories
2. OSGi bundles
3. RAR archives
4. Deployment plan resources
5. EJB archives
6. WAR archives
7. EAR archives

Note

For each category, file names are chosen in alphabetical order

Then this directory is periodically polled in order to deploy new archives. For more information have a look at the depmonitor service configuration [configuration_guide.html#services.depmonitor.config]

- the `lib/` directory [2]

  Used for extending class loaders. It contains one sub directory:

directory	description
`ext`	For non-bundle extensions

- the `logs/` directory

  where the log files are created at run-time

- the `work/` directory

  a working directory for JOnAS

- the `repositories/` directory

  this sub tree contains the following repositories used to store OSGi bundles and Java EE applications. Archives located in these repositories are priority in case they are also located in JONAS_ROOT/repositories

  - `maven2-internal/` this directory is created during the building process of a JONAS_BASE environment. It may contain the JORAM resource adapter for JOnAS. It is used for internal purpose and should not be modified. This directory is structured as a Maven2 repository.
  - `<repository-id>/` contains archives downloaded through deployment plans from this repository.

2.1.3. JONAS_BASE creation

1. To create a JONAS_BASE template from scratch :

   Unix

```
export JONAS_BASE=~/my_jonas_base
cd $JONAS_ROOT/templates/newjb
ant -f build-jb.xml create_jonas_base
```

Windows

```
set JONAS_BASE=my_jonas_base
cd %JONAS_ROOT%/templates/newjb
ant -f build-jb.xml create_jonas_base
```

This will copy all the required files and create all the needed directories.

2. Another way to create a JONAS_BASE template from scratch :

   `$JONAS_ROOT/bin` must be set in the system path:

   Unix

   ```
   export JONAS_BASE=~/my_jonas_base
   newjb
   ```

   Windows

   ```
   set JONAS_BASE=my_jonas_base
   newjb
   ```

   The JONAS_BASE content created with the **newjb** command is well suited to run the JOnAS JEE conformance test suite and the example applications without any additional configuration.

   In order to customize a JONAS_BASE with specific property values (port numbers, services, protocols etc...), you must edit the `$JONAS_ROOT/templates/newjb/build-jb.properties` file or `$HOME/jb.config/conf/jonas-newjb.properties` file before running **newjb**.

   For further customization that cannot be performed by **newjb** you should modify the generated files in `$JONAS_BASE/conf`. For more information see the description of the newjb command in Commands Reference Guide [./command_guide.html#commands.newjb].

2.1.4. JONAS_BASE/conf description

This directory contains configuration files in various format (properties files, xml files).

The main configuration file is *jonas.properties* but there are also:

- Templates for configuring access to databases for the *dbm* service, (Oracle, PostgreSQL, Sybase, DB2, MySQL, HSQLDB, InterBase, FirebirdSQL, Mckoi SQL, InstantDB) respectively in *Oracle1.properties*, *PostgreSQL1.properties*,etc... All these databases have been tested with JOnAS.
- Mail resources templates : *MailMimePartDS1.properties*, *MailSession1.properties*
- JORAM configuration files : *a3debug.cfg*, *a3servers.xml*, *joramAdmin.xml*
- EasyBeans ejb3 container configuration file is named *easybeans-jonas.xml.*
- *carol.properties*, *jacorb.properties* for configuring the RMI implementation used through CAROL.
- Configuration files for clustering : *cmi-config.xml*, *clusterd.xml*, *domain.xml*, *jgroups-discovery.xml*, *jgroups-ha.xml*, *jgroups-cmi.xml*.
- Configuration files related to security: *jaas.config*, *java.policy*, *jonas-realm.xml*

- Web container configuration files:
  - *tomcat6-server.xml*, *tomcat6-context.xml*, *tomcat6-web.xml* for Tomcat,
  - *jetty6.xml jetty6-web.xml* for Jetty.
- Web services configuration files:*uddi.properties, file1.properties.*
- Client container configuration file: *jonas-client.properties*
- JOnAS traces configurations files: *trace.properties, traceclient.properties*
- Transaction recovery configuration file : *jotm.properties*
- P6Spy options file: *spy.properties*
- Java Service Wrapper configuration file: *wrapper.conf*
- Deployment plan initial repositories are stored in `initial-repositories.xml` file.
- *jmx.access*, *jmx.passwords* and *jmx.rolbased.access* are configuration files used to secure the JMX connector access.
- *jndi-interceptors.xml* is used by the JNDI Interceptors allowing for example to automatically close the JDBC connections if they're not closed by the application.
- *classloader-default-filtering.xml* allows to hide to applications some classes exported by the Application Server.
- *banner.txt* allows to change the banner of the JOnAS scripts.

Most of these files are described in following sections.

2.1.5. Server and services configuration

`$JONAS_BASE/conf/jonas.properties` is the key file for configuring JOnAS.

This file is used for:

- setting some global properties for the JOnAS instance
- choosing the list of JOnAS services to be launched at startup
- customizing each services

2.1.5.1. Global properties

```
#  Name of the JOnAS server
#  default value is "jonas"
jonas.name    jonas

#  Name of the JOnAS domain
#  default value is "jonas"
domain.name    jonas

#  Enable the Security context propagation (for jrmp)
jonas.security.propagation    true

#  Enable the Security manager
#  default value is true (if not set)
#  Setting this to false implies a collocated registry and setting in carol.properties:
#  carol.jvm.rmi.local.registry=true
jonas.security.manager    false
```

```
#  Enable csiv2
jonas.csiv2.propagation    true

#  Enable the Transaction context propagation
jonas.transaction.propagation    true

#  Set the name of log configuration file
jonas.log.configfile    trace

#  Set to true if the server is a master
jonas.master    false

#  Set to true in order to execute the JOnAS Server in development mode.
#
#  WAR archive deployment case in development mode (for single or EAR packaged WARs):
#  Each modified WAR archive will be unpacked in the working directory of the JOnAS Server
#  in a different folder to avoid file locks. This is especially useful in a Windows
 environment.
jonas.development    true
```

Note

setting jonas.security.manager to `false` implies a colocated registry and implies to set in `carol.properties`:

```
carol.jvm.rmi.local.registry=true
```

2.1.5.2. List of JOnAS services

Here is the list of default services activated at starting time:

```
jonas.services    jtm, db, security, resource, ejb3, jaxws, web, ear, depmonitor
```

The possible services are:

```
registry, jmx, security, jtm, db, mail, wc, dbm, wm, resource, cmi, ha, versioning, ejb2,
 ejb3, jaxrpc, jaxws, web, ear, depmonitor, discovery, resourcemonitor, smartclient, wsdl-
publisher
```

cmi	this service provides support for the clustering of RMI objects.
db	this service is used for launching a Java database implementation. By default, HSQLDB java database is used.
dbm	this service is needed by application components that require access to one or several relational databases. It may be an alternative to the usage of a JDBC resource adapter via the resource service.
depmonitor	this service is used to control the application's deployment process in JOnAS.
discovery	this service allows dynamic administration of management domains.
ear	this service provides support for Java EE applications (`.ear` files).
ejb2	this service provides support for EJB 2.x components (EjbJars).
ejb3	this service provides supports for EJB 3.0 components (EjbJars).
ha	this service provides high-availability replication mechanims for stateful session beans (EJB 2.x only).
jaxrpc	this service provides support for JAX-RPC 1.1 webservices (J2EE 1.4 style, based on `webservices.xml` deployment descriptors).

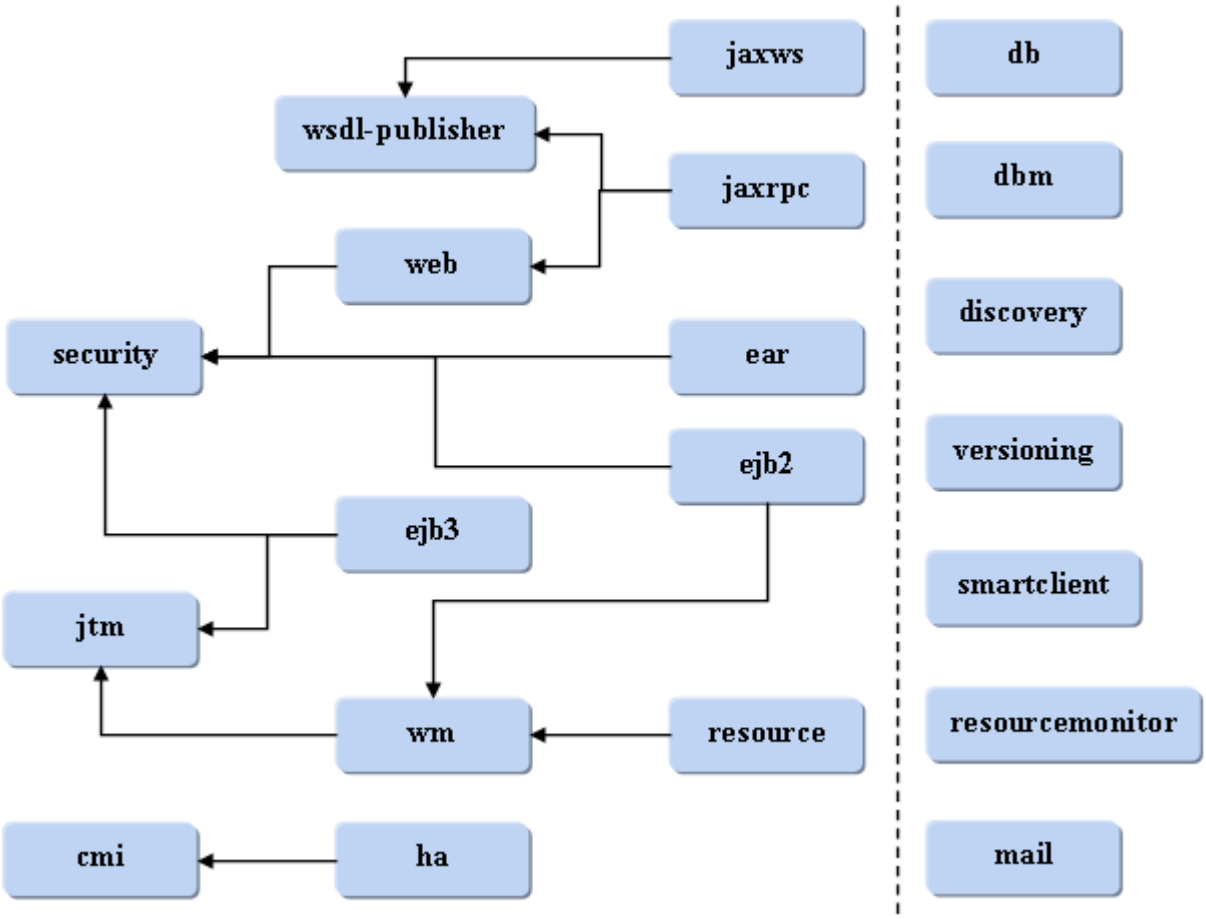
jaxws	this service provides support for JAX-WS 2.0 webservices (Java EE 5.0 style based on JWS annotations).
jmx	this service is needed in order to administrate the JOnAS servers and the JOnAS services via a JMX-based administration console. It is automatically launched before all the other services when starting JOnAS.
jtm	this service provides support of distributed transactions management.
mail	this service is required by applications that need to send e-mail messages.
registry	this service is used for binding remote objects and resources that will later be accessed via JNDI. It is automatically launched before all the other services when starting JOnAS.
resource	this service provides support for resource adapters conformant to the Java EE Connector Architecture Specification.
resourcemonitor	this service is related to the deployment plans. It allows to reload resources deployed through deployment plans by checking periodically repositories where are located original resources.
smartclient	this service lets remote clients download classes and other resources necessary for connecting to JOnAS services (JNDI context factories, EJB3 interceptors, ...) directly from the JOnAS server they're dealing with.
security	this service is needed for enforcing security at runtime.
versioning	this service has been designed for dynamic redeployment of applications,without any application downtime and without users' sessions being lost.
wc	this service cleans up periodically the work directory of the JOnAS server.
web	this service provides support for web components (as Servlets and JSP). JOnAS provides two implementations of this service, one based on Tomcat and another on Jetty.
wm	this service provides a JCA WorkManager implementation (offering a manageable Thread Pool for resource adapters components).
wsdl-publisher	this service provides an alternate WSDL publishing mechanism (compared to the usual URL based publishing).

2.1.5.3. Service startup policies

JOnAS will try to start declared services in the order in which they appear in the list except for the *depmonitor* service which is always started at the end. If some services require other ones (even non declared in the list), service requirements will be started first.

To simplify the declaration of JOnAS services and to ensure that all service requirements are fulfilled, some services declare explicitly their dependencies in order to start them automatically.

The picture below describes mandatory dependencies beetwen JOnAS services. A link between two services means that a service requires another one. Note on the right side services without dependency links.



Caution

Optional service dependencies are not described in this picture. They have to be declared in the list of JOnAS services when required.

- *registry* and *jmx* services can be omitted from the list because they are automatically launched.
- As an example, starting the *web* service involves the startup of the *security* service. Declaring the *web* service in the list of JOnAS services without declaring the *security* service may be a solution.
- As an example, starting the *resource* service involves the startup of the *wm* and the *jtm* services. Declaring the *resource* service in the list of JOnAS services without declaring the *wm* and the *jtm* service may be a solution.

2.1.5.4. Customizing services

Configuration parameters for services follow a strict naming convention: a service **XX** will be configured via a set of properties:

jonas.service.XX.foo something

jonas.service.XX.bar else

each service **XX** must contain the property jonas.service.XX.class indicating the name of the java class that implements the service:

jonas.service.XX.class aa.bb.XXImpl

This allow experimented user to replace built-in service by an alternative implementation.

For example here is the part of `jonas.properties` file related to the customization of the **jtm** service:

```
###################### JOnAS JTM Transaction service configuration
#  Set the name of the implementation class of the jtm service
jonas.service.jtm.class    org.ow2.jonas.tm.jotm.JOTMTransactionService

#  Set the Transaction Manager launching mode.
#  If set to 'true', TM is remote: TM must be already launched in an other JVM.
#  If set to 'false', TM is local: TM is going to run into the same JVM
#  than the jonas Server.
jonas.service.jtm.remote    false

#  Set the default transaction timeout, in seconds.
jonas.service.jtm.timeout    60
```

see Section 2.4, “Configuring JOnAS Services” for a complete description of the services configuration.

2.1.5.5. Development vs Production mode

JOnAS may be configured to be in development mode and in production mode. This can be defined by setting the **jonas.development** global property. Activating one of this mode changes some server behaviours as described in the following section.

2.1.5.5.1. Development mode

This is the default mode.

- Starts automatically the workcleaner (*wc*) service.
- Allows to start automatically services which are required to initiate the deployment of the Java EE archives. Ex: deploying the sample.war will trigger the startup of the *web* service.

  List of services that may be started dynamically depending on the deployed Java EE archives:

  ```
  web (WARs), ejb2 (EJB2s), ejb3 (EJB3s), resource (RARs), ear (EARs)
  ```

  Note

  Known limitations: if an application needs some additional services to work like for example Web Services support (*jarxrpc* or *jaxws* services), the administrator have to add those kind of services manually in the static description of JOnAS services.

- In case of the development property of the *depmonitor* service is set to inherit, the period scan of directories managed by the *depmonitor* (by default the `deploy/` directory) service will be enabled.
- WAR archive deployment case (for single or EAR packaged WARs). Each WAR archive is unpacked in the working directory of the JOnAS server in a different folder to avoid file locks. This is especially useful in a Windows environment.
- Necessary to enable the onDemand feature of the *web* service.

2.1.5.5.2. Production mode

This mode is recommended in industrial production context.

- In case of the development property of the depmonitor service is set to inherit, the period scan of directories managed by the depmonitor (by default the `deploy/` directory) service will be disabled.

- WAR archive deployment case (for single or EAR packaged WARs). Each WAR archive is unpacked in the working directory of the JOnAS server in the same folder.
- Force the disabling of the onDemand feature of the *web* service.

2.2. Configuring the communication protocol and JNDI

JOnAS provides a multi-protocol support through the integration of the CAROL component.

Supported communication protocols are the following:

- RMI/JRMP is the JRE implementation of RMI on the JRMP protocol. This is the default communication protocol.
- RMI/IIOP is the JacORB [http://www.jacorb.org/] implementation of RMI over the IIOP protocol.
- IRMI is an RMI implementation that can be used with Open Source JDK that doesn't provide `com.sun.*` classes.

For each of these protocols, the clustering of RMI objects can be enabled with the component CMI.

2.2.1. Choosing the Protocol

The choice of the protocol is made in the carol.protocols property of `carol.properties` file in `JONAS_BASE/conf` directory.

```
carol.protocols=jrmp
```

2.2.1.1. configuring jrmp protocol

```
carol.protocols=jrmp 1
carol.jrmp.url=rmi://localhost:1099 2
carol.jvm.rmi.local.call=false 3
carol.jvm.rmi.local.registry=false 4
carol.jrmp.server.port=0 5
carol.jrmp.interfaces.bind.single=false 6
```

1. choice of the protocol or list of protocols
2. connexion url to the RMI registry the hostname (localhost) and port number must be changed if needed. In a distributed configuration changing the hostname is mandatory.
3. if true local calls are optimized: calls to methods of the remote interface are treated as call to local methods (it is not always possible depending on the packaging of the application).
4. if true a local Naming context is used. This must be used only with a collocated registry and it is mandatory when the jonas.security.manager property of jonas.properties is set to true.
5. exported objects will listen on this port for remote method invocations. 0 means random port. Specify a port may be useful when the server run behind a firewall.
6. if true use only a single interface (choosen from the url) when creating the registry. False means use all interfaces available.

2.2.1.2. configuring RMI/IIOP protocol

The JacORB implementation of RMI over the IIOP is used. The configuration file of JacORB is the `$JONAS_BASE/conf/jacorb.properties` file.

As for the other protocols RMI over IIOP is ready to used in the default distribution. It is only for tuning purpose that the `$JONAS_BASE/conf/jacorb.properties` file must be customized.

By default the CORBA Naming service is run using the port 2001 (as it is set in the `carol.properties` file)

So the only thing to do for working in RMI over IIOP is to set the property protocols in carol.properties:

```
carol.protocols=iiop
# RMI IIOP URL
carol.iiop.url=iiop://localhost:2001
carol.iiop.server.port=0 1
carol.iiop.server.sslport=2003 2
carol.iiop.PortableRemoteObjectClass=org.ow2.jonas.registry.carol.delegate.JacORBPRODelegate
  3
```

1 0 means random port
2 this port is used only if SSL mode is enabled (default configuration = not used).Is used to set the SSL port of the objects listener
3 delegate used by JOnAS for rmi-iiop protocol.

2.2.1.3. configuring irmi protocol

```
carol.protocols=irmi
carol.irmi.url=rmi://localhost:1098 1
carol.irmi.server.port=0 2
carol.irmi.interfaces.bind.single=false 3
```

1 for irmi the default port is 1098
2 exported objects will listen on this port for remote method invocations:0 means random port.

Caution

if the port is set to n the port 'n + 1' will be used by the JMX server.So, for the firewall configuration, you have to open the port numbers 'n' and 'n+1'

3 if true use only a single interface when creating the registry (specified in carol.irmi.url property). Default configuration = false(use all interfaces available)

2.2.1.4. enabling clustering of RMI objects

CMI is the component to use for clustering purpose. It is embedded in the component CAROL.

CMI is just composed of wrappers and interceptors and is fully independant of the implementation of protocol. CMI relies on JGroups [http://www.jgroups.org/javagroupsnew/docs/index.html] group-communication protocol for ensuring the replication of the cluster view. CMI provides jndi high availability, the load-balancing and fail-over at the EJB level.

For using CMI with a protocol (in addition to the activation of service cmi), a property must be added in `carol.properties`:

```
carol.jrmp.cmi=true 1
carol.iiop.cmi=true
carol.irmi.cmi=false 2
```

1 Enable clustering with jrmp
2 Disable clustering with imi

Note

By default, the property is set at true.

2.2.1.5. multi protocol configuration

JOnAS can be configured to use several protocols simultaneously. To do this, just specify a comma-separated list of protocols in the carol.protocols property of the `carol.properties` file. For example:

```
carol.protocols=iiop,jrmp
```

```
carol.jrmp.url=rmi://localhost:1099
carol.iiop.url=iiop://localhost:2001
```

Caution

When iiop is used in a multiprotocol configuration, it must appear at the first position in the protocol list.

2.3. Configuring the logging System

Monolog [http://monolog.objectweb.org/doc/index.html] is the Objectweb solution for logging. It is not only a new logging implementation but can be seen as a bridge between different logging implementations. A library that uses the Monolog API can be used with any logging implementation at runtime.

Furthermore some components of JOnAS like CAROL, JOTM, Tomcat etc... doesn't use the Monolog API but Jakarta commons loggins or log4j or other implementation. However all these components will be configured via the JOnAS Monolog configuration file.

2.3.1. Monolog

JOnAS Monolog configuration files are:

- `$JONAS_BASE/conf/trace.properties` [3]

  which is the server side Monolog configuration file

- `$JONAS_BASE/conf/traceclient.properties`

  which is used for a client in a client container.

Configuring trace messages inside JOnAS can be done in two ways:

1. Changing the `trace.properties` file to configure the traces statically, before the JOnAS Server is run
2. Using the jonas admin command or the JonasAdmin administration tool to configure the traces dynamically while the JOnAS Server is running. In this case the modification are not persistent (`trace.properties` file is not modified).

2.3.2. trace.properties syntax

Applications make logging calls on *logger* objects.Loggers are organized in a hierarchical namespace and child *loggers* may inherit some logging properties from their parents in the namespace. *Loggers* allocates messages and passes them to *handler* for output; they uses logging *levels* in order to decide if they are interested in by a particular message.

In `trace.properties` it is possible to define *handlers*, *loggers*, *levels*:

- handlers

  A handler represents an output, is identified by its name, has a type, and has some additional properties. By default three handlers are used:

  - **tty** is basic standard output on a console
  - **logf** is a handler for printing messages on a file
  - **mesonly** handler used by generation tools for traces without header

Each handler can define the header it will use, the type of logging (console, file, rolling file), and the file name.

The handler properties are the following:

- type: is the type of the handler that may be:
  - Console : Log stream ends inside `System.out` or `System.err`
  - File : Log stream is directed into a file
  - Rollingfile : A file set is used to roll the logs
  - JMX : Logging actions are send to the JMX notification system
- pattern:is the message format. A pattern can be composed of elements. An element is prefixed by the % character.The possible items:
  - **%h**: the thread name
  - **%O{1}** : the Class name (basename only)
  - **%M** the method name
  - **%L** the line number
  - **%d** the date
  - **%l** the level
  - **%m** the message itself
  - **%n** a new line
- output: is the output filename.

  If *automatic*[4] is used, JOnAS will replace this tag with a file pointing to `$JONAS_BASE/logs/<jonas_name_server>-<timestamp>.log`

  Switch is used for logging either on `System.out` or `System.err` depending on the level of the log
- fileNumber: is the number of file to use (for RollingFile)
- maxSize:is the maximal size of the file (for Rolling file)

Note that another handler, named **jmxHandler**, can be used to allow to view the recent logs in the JOnAS administration console. By default the definition of this handler is commented, for performance reason.

- loggers

  Loggers are identified by names that are structured as a tree. The root of the tree is named *root*. Each logger is associated with a topic. Topic names are usually based on the package name. Each logger can define the handler it will use and the trace level (see below). By default loggers inherit their level from their parents.

  By default handlers assigned to the parent logger are automatically assign to child loggers. Setting 'additivity' to false inform the system that the logger will use only its own set of handlers.[5]

- levels

the trace levels are the following:

- ERROR errors. Should always be printed.
- WARN warning. Should be printed.
- INFO informative messages.
- DEBUG debug messages. Should be printed only for debugging.

2.3.3. default trace.properties file

```
log.config.classname org.objectweb.util.monolog.wrapper.javaLog.LoggerFactory  1

handler.tty.type Console 2
handler.tty.output Switch 3
handler.tty.pattern  %d : %O{1}.%M : %m%n 4

handler.logf.type File 5
handler.logf.output automatic 6
handler.logf.pattern  %d : %l : %h : %O{1}.%M :    %m%n

logger.root.handler.0 tty 7
logger.root.handler.1 logf 8

logger.root.level INFO 9
logger.org.objectweb.level INFO
logger.org.ow2.level INFO

#logger.org.ow2.jonas.lib.ejb21.level DEBUG 10

handler.mesonly.type Console 11
handler.mesonly.output Switch
handler.mesonly.pattern  %m%n

logger.org.ow2.jonas.generators.genic.handler.0 mesonly 12
logger.org.ow2.jonas.generators.genic.additivity false 13

[...]
```

1. Definition of the wrapper to use: here the java logging API wrapper.
2. Definition of the console handler tty
2. Switch means that the logs will be on System.out or System.err depending of the level of the log.
4. Definition of the message format. here it contains the date followed by ':' the basename of the class followed by '.' the method name followed by ':' the message itself terminated by newline.
5. Definition of the file handler logf
6. Logs are in a file whose name is `$JONAS_BASE/logs/<jonas_name_server>-<timestamp>.log`
7. Definition of the root logger. It uses handler tty
8. Definition of the root logger: It uses also handler logf
9. Definition of the root logger: level INFO is used for all child loggers if there is no overriden definition
10. This line must be uncommented for setting DEBUG level for the logger used in the jonas ejb21 module
11. Definition of the console handler mesonly used by generator tool, such as GenIC, which want to log messages without headers
12. Definition of the handler used by the logger org.ow2.jonas.generators
13. This logger wants to use its own handler.

2.3.4. Tips for setting loggers for JOnAS

When a problem occurs it may be worth to set some debugging traces in the JOnAS server. It is not easy to know which logger to set to obtain the pertinent traces that may help the debbugging process.

The `trace.properties` file contains several commented lines prepared to set loggers in DEBUG level.

Usually the name of loggers are related to the java package name in which it is used.

- To set debug traces of the EJB2 container uncomment one or more lines related to logger org.ow2.jonas.lib.ejb2 for example:

```
logger.org.ow2.jonas.lib.ejb21.interp.level DEBUG
logger.org.ow2.jonas.lib.ejb21.synchro.level  DEBUG
logger.org.ow2.jonas.lib.ejb21.tx.level DEBUG
```

- To set traces related to resource adapters:

```
logger.org.ow2.jonas.jca.level DEBUG
logger.org.ow2.jonas.jca.pool.level DEBUG
```

- To set traces into the CAROL library::

```
logger.org.ow2.carol.level DEBUG
```

- To set traces in JORAM:

```
logger.fr.dyade.aaa.level DEBUG  (for the MOM)

# for the JORAM resource adapter:
logger.org.objectweb.joram.client.jms.Client.level DEBUG
logger.org.objectweb.joram.client.connector.Adapter.level DEBUG
```

- To set traces in Tomcat:
  - for all web application :

    ```
    logger.org.apache.catalina.core.ContainerBase.[jonas].[localhost].level DEBUG
    ```

    jonas is the attribute name of the element Engine in $JONAS_BASE/conf/tomcat6-server.xml

    localhost is the attribute name of the element Host in $JONAS_BASE/conf/tomcat6-server.xml

  - for a particular web application :

    ```
    logger.org.apache.catalina.core.containerBase.[jonas].[localhost].[jonasAdmin].level
     DEBUG
    ```

    jonas is the attribute name of the element Engine in $JONAS_BASE/conf/tomcat6-server.xml

    localhost is the attribute name of the element host in $JONAS_BASE/conf/tomcat6-server.xml

    jonasAdmin is the name of the web application

Note

the attributes debug in elements of $JONAS_BASE/conf/tomcat6-server.xml are not used anymore in Tomcat.

- There are a lot of traces possible for management, discovery, jtm, clustering, mail, ear,...

2.3.5. Logging with particular log systems

2.3.5.1. java logging API

If Monolog is configured to use the JDK logger it will replace the JDK logger implementation with its own implementation and so all JDK logs are intercepted by Monolog. By default Monolog is configured to use the JDK logger.

2.3.5.2. Jakarta commons logging

There is no special configuration file for Jakarta commons login.If it is used on top of the java logging API it is the same case than the previous section.

2.3.5.3. log4j

JOnAS don't provide the corresponding `jar` file so, log4j must be packaged (`.jar` file and `log4j.properties`) in any application that want to use it. The l`og4j.properties` file must be configured correctly.

If log4j is used by several applications it is possible to centralize the log4j configuration by putting log4j.properties in `$JONAS_BASE/conf` and log4j jar file in `$JONAS_BASE/lib/commons`.

2.4. Configuring JOnAS Services

Here is the list of possible services

```
registry,jmx,security,jtm,db,mail,wc,dbm,wm,resource,cmi,ha,versioning,ejb2,ejb3,jaxrpc,jaxws,web,ear,depmonito
publisher
```

In this chapter we will describe how to configure each service in the `jonas.properties` file.

2.4.1. cmi service configuration

The configuration of the **cmi** service is available through the file `$JONAS_BASE/conf/cmi-config.xml`.

The CMI service can be configured in two modes:

- *server mode* with a cluster view manager created locally, i.e. with a local instance of a replicated CMI registry.
- *client mode* without a local cluster view manager, in this case a list of providers urls (i.e. a list of cluster view manager urls) is given for accessing to the remote CMI registries.

The *server mode* is simpler to configure, the *client mode* requires to define statically a list of providers urls. The *server mode* starts a Group Communication Protocol instance (e.g. JGroups) and thus increases the resources consumption compare to the *client mode*.

Note

The CMI configuration file may contain two parts: a `server` element which corresponds to the server mode configuration and a `client` element for the client mode configuration. If the two are present, only the `server` element is loaded which means that the server mode is configured.

2.4.1.1. Server mode configuration

The server element contains the following elements:

Example 2.1. Configuring the cmi service in the server mode

```
<cmi xmlns="http://org.ow2.cmi.controller.common"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:jgroups="http://org.ow2.cmi.controller.server.impl.jgroups">
    <server>
        <jndi> 1
            <protocol name="jrmp" noCmi="false" />
        </jndi>
        <viewManager  2
            class="org.ow2.cmi.controller.server.impl.jgroups.JGroupsClusterViewManager"> 3
            <jgroups:config 4
                    delayToRefresh="60000" 5
                    loadFactor="100" 6
                    confFileName="jgroups-cmi.xml" 7
                    recoTimeout="30000" 8
                    groupName="G1"> 9
                <components> 10
                 <event />
                </components>
            </jgroups:config>
        </viewManager>
    </server>
</cmi>
```

1. `jndi` element - optional. Enable to specify that a protocol must not be clustered with CMI (administration uses, ...). Here, the clustering of `jrmp` protocol can be disabled by setting `true` to the `noCmi` attribute.
2. `viewManager` element - mandatory. Defines the view manager configuration (registry replication, refresh time, ...).
3. `class` attribute - mandatory. Specifies the protocol implementation to use for replicating the view (CMI registry). Here the JGroups implementation is set.
4. `jgroups:config` element - mandatory. Define the JGroups related parameters.
5. `delayToRefresh` attribute - optional. Refresh period of the client view (in ms). For example, it expresses the maximum delay for taking into account a load-balancing parameter update.
6. `loadFactor` attribute - optional. Specifies the initial load-factor of the current node used in the weigthed round robin policy.
7. `confFileName` attribute - mandatory. Specifies the JGroups's stack configuration filename (found in the `$JONAS_BASE/conf` directory).
8. `recoTimeout` attribute - optional. Specifies the reconnection timeout after a shunning or an error in the group communication protocol (in ms). If the timer expires, an exception is thrown.
9. `groupName` attribute - mandatory. Specifies the JGroups channel name used by the CMI cluster view replication mechanism.
10. `components` element - mandatory. Enable the events component into CMI. This element must not be modified.

Note

Refer to the clustering guide [clustering_guide.html#faq.jgroups] for issues related to JGroups.

2.4.1.2. Client mode configuration

The client element contains the following elements:

Example 2.2. Configuring the cmi service in the client mode

```
<cmi xmlns="http://org.ow2.cmi.controller.common"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <client noCmi="false"> 1
     <jndi> 2
         <protocol name="jrmp">
             <providerUrls>
                 <providerUrl>rmi://localhost:1099</providerUrl>
                 <providerUrl>rmi://localhost:2001</providerUrl>
             </providerUrls>
         </protocol>
     </jndi>
    </client>
</cmi>
```

1 `noCmi` attribute - optional. Enable to specify that CMI must be disabled.
2 `jndi` element - mandatory. Specify a list of providers URLs for a given protocol. It is not necessary to set the whole list of cluster members, a subset is enough. However for ensuring high availability, at least two providers URLs must be mentionned.

2.4.2. db service configuration

The **db** service is an optional service that can be used to start a java database server in the same JVM as JOnAS.

By default the database used is HSQLDB. [http://hsqldb.org/]

Here is the part of `jonas.properties` related to **db** service:

```
###################### JOnAS DB service configuration
#
#  Sets the name of the implementation class of the db service (hsql for example)
jonas.service.db.class   org.ow2.jonas.db.hsqldb.HsqlDBServiceImpl
jonas.service.db.port    9001
jonas.service.db.dbname  db_jonas
jonas.service.db.users   jonas:jonas

# Multiple users
#jonas.service.db.users  jonas:jonas,login:password
```

Here it is possible to customize :

- the listening port
- the database name
- By default, the user is named jonas with the password jonas. In order to add new users, the property jonas.service.db.users needs to be updated by using a comma separated list as follow:

```
jonas.service.db.users   login:password,anotherlogin:password
```

The database may be used by Java EE component via JDBC resource adapters or via the **dbm** service. For the former case the same information (listening port, database name, login,password) must appear in the JOnAS connector deployment descriptor, in the latter they appear in the `$JONAS_BASE/conf/HSQLDB1.properties.` So, if these previous properties must be changed in jonas.properties, they must be also changed in these files.

The **db** service has been provided in the jonas distribution mainly to run easily the JOnAS exemple, without having to set a database first. For most usages, the JOnAS users should remove

it from the list of services and remove also HSQL1 from jonas.service.dbm.datasources property in `$JONAS_BASE/conf/jonas.properties` file.

For users that choose HSQLDB as database it is highly recommended to refer to the Hsqldb User Guide [http://hsqldb.org/web/hsqlDocsFrame.html]. It is worth to note that the default configuration file used by HSQLDB server can be found in `$JONAS_BASE/work/hsqldb/jonas/db_jonas.properties` directory.

In order to launch several HSQLDB instances, the configuration needs to be duplicated and the new configuration will be prefixed by jonas.service.<mynewdbservice>.

2.4.3. depmonitor service configuration

The **depmonitor** service scans periodically some directories in the aim of deploying J2EE applications or OSGi bundles on a JOnAS server. By default, you have to put the application files into the `$JONAS_BASE/deploy` directory in order to deploy them. It is possible to parse other directories by setting the directories property in the service configuration.

The development attribute in the configuration allows to choose if the **depmonitor** service is in development mode or not :

- The deployment monitor can be configured to detect at runtime if an application is added, removed or changed to respectively deploy it, undeploy it or redeploy it. This functionnality can be useful during the development phase.
- For a prodution usage of the JOnAS server, this functionnality can be disabled so that the application files will be deployed only at startup. In this configuration, the jonas admin command [command_guide.html#commands.jonas.jonasadmin] or the jonasAdmin user interface will be used to perform deployment actions.

As the parsed directories may contain files that must not be deployed, a list of file exclusions can be defined.

```
###################### JOnAS Deployment Monitor
#
# Set the name of the implementation class of the depmonitor service
#
jonas.service.depmonitor.class
 org.ow2.jonas.deployablemonitor.DeployableMonitorService

# Set the execution mode (three possible values):
# - inherit: inherit of the value of the "jonas.development" property
# - true   : development mode
# - false  : production mode
jonas.service.depmonitor.development    inherit                                    1

jonas.service.depmonitor.directories                                               2
# List (comma separated) of exclusion patterns (based on names, not directories)
jonas.service.depmonitor.exclusions     README                                     3

# Monitor interval in milliseconds
jonas.service.depmonitor.monitorInterval    5000                                   4
```

1. If the property value is true, the directories are parsed periodically to detect file addition, modification or deletion. Else, the directories are parsed only at startup
2. A comma-separated list of directories which contain files to deploy
3. A comma-separated list of file names to exclude
4. Monitor interval in milliseconds between two scans

2.4.4. dbm service configuration

The **dbm** service (database manager service) allow access to one or more relational databases. It will create and use `DataSource` objects. Such a DataSource object must be configured according to the database that will be used for the persistence of a bean.

Caution

the recommended way to access to databases is to use the **resource** service deploying JDBC resource adapter

The **dbm** service provides a generic driver-wrapper that emulates the `XADataSource` interface on a regular JDBC driver.It is important to note that this driver-wrapper does not ensure a real two-phase commit for distributed database transactions. When it is necessary to use a JDBC2-XA-compliant driver access to the databases must be done via a JDBC resource adapter XA compliant (more information can be found in Section 2.6, “Configuring JDBC Resource Adapters”

Here is the part of `jonas.properties` related to **dbm** service:

```
###################### JOnAS DBM Database service configuration
#
#  Set the name of the implementation class of the dbm service
jonas.service.dbm.class    org.ow2.jonas.dbm.internal.JOnASDataBaseManagerService

#  Set the jonas DataSources. This enables the JOnAS server to load
#  the data dources, to load related jdbc drivers, and to register the data
#  sources into JNDI.
#  This property is set with a coma-separated list of Datasource properties
#  file names (without the '.properties' suffix).
#  Ex: Oracle1,InstantDB1 (while the Datasources properties file names are
#                          Oracle1.properties and InstantDB1.properties)
jonas.service.dbm.datasources    HSQL1
```

For the **dbm** service it is possible to:

- set a list of datasource names via property jonas.service.dbm.datasources.

  for each name XX appearing in this list a `XX.properties` file must exist in `$JONAS_BASE/conf`

Access to a particular database via **dbm** service is configured in datasource.properties files that must be located in `$JONAS_BASE/conf.`

2.4.4.1. Datasource.properties files

In the JOnAS distribution several templates of datasource.properties files are provided one for Oracle, PostgreSQL, Sybase, DB2, MySQL, HSQLDB, InterBase, FirebirdSQL, Mckoi SQL, InstantDB) respectively in *Oracle1.properties*, *PostgreSQL1.properties* etc...

A complete description of the datasource.properies file can be found inSection 2.8, “Configuring JDBC DataSources”

2.4.5. discovery service configuration

The role of the **discovery** service is to enable dynamic **domain management**. Recall that domain management means management of all the servers running in the domain, from the common administration point represented by a master server.

The discovery service allows a master to detect servers starting and stopping in the domain. Moreover, a master can discover servers there were already running in the domain when it started.

The discovery service implements a *greeting* mechanism to enforce servers' name unicity in the domain. This mechanism prevents starting a new server in the domain, if a server having the same name is already running in the domain.

There are two available implementations for the discovery service: one based on IP multicast, the other based on JGroups. The former, introduced in JOnAS 4, is deprecated. The latter, has the advantage to allow for cluster daemons detection.

All servers and in the domain must choose the same implementation. The choice is made upon the implementation class name:

```
###################### JOnAS Discovery service
#
#  Set the name of the implementation class and initialization parameters
# JGropus implementation
jonas.service.discovery.class=org.ow2.jonas.discovery.jgroups.JgroupsDiscoveryServiceImpl
# Uncomment this to set Multicast implementation
#jonas.service.discovery.class=org.ow2.jonas.discovery.internal.MulticastDiscoveryServiceImpl
```

2.4.5.1. Configuration for IP multicast based implementation

You have to provide initialization parameters in jonas.properties file for:

- Multicast address and port. These must be identical for all servers in the domain. Use properties:
  - jonas.service.discovery.multicast.address
  - jonas.service.discovery.multicast.port

  beware that multicast adresses must be consequently allocated through the network.
- The time-to-live for packets: use property:
  - jonas.service.discovery.ttl

  this parameter indicates the number of gateway hops for packets.
  - if ttl = 0 the discovery scope is the host (multicast packet aren't routed to network interfaces).
  - if ttl =1 the discovery scope is limited to the subnetworks the host is attached to (multicast packets cross the network interfaces but will be discarded by the next gateway).
  - if ttl =N>1 the discovery packets may cross N-1 gateways (provided that these gateways are configured to propagate multicast packets).
- In the case of a master server, the jonas.service.discovery.source.port property must be set with an available port number.
- The greeting mechanism. Use properties:
  - jonas.service.discovery.greeting.port
  - jonas.service.discovery.greeting.timeout

  Note that two servers on the same host must have different values in greeting.port property.

Example:

```
jonas.service.discovery.multicast.address=224.224.224.224
jonas.service.discovery.multicast.port=9080
jonas.service.discovery.ttl=1
#  For a master server, configure the client source port with this property
jonas.service.discovery.source.port=9888

#  A multicast greeting message is sent out when discovery service is started.
#  The starting server listens at the port jonas.service.discovery.greeting.port
#  (default 9899) for a response for jonas.service.discovery.greeting.timeout miliseconds
#  (default 1000 ms). If a pre-existing server has the same server name as this one,
#  this server's discovery service will be terminated.
jonas.service.discovery.greeting.port=9899
jonas.service.discovery.greeting.timeout=1000
```

2.4.5.2. Configuration for JGroups based implementation

JGropups configuration being more complex, a specific configuration file have to be used. The name of this file is given by the jonas.service.discovery.jgroups.conf property. Two other properties have to be initialized:

- The name of the JGroups group used by the the discovery service to exchange messages.
  - jonas.service.discovery.group.name
- The reconnection timeout for the JGroups channel.
  - jonas.service.discovery.group.reconnection.timeout

Example:

```
jonas.service.discovery.jgroups.conf=jgroups-discovery.xml
jonas.service.discovery.group.name=JGroupsDiscovery
jonas.service.discovery.group.reconnection.timeout=5000
```

You can find in JOnAS distribution, under `JONAS_ROOT/conf`, a `jgroups-discovery.xml` file. This file containes a JGroups stack configuration for the UDP protocol.

Note

Refer to the clustering guide [clustering_guide.html#faq.jgroups] for issues related to JGroups.

2.4.5.3. Cluster deamon configuration for discovery

In order to be detected by a master server, a cluster daemon has to be properly configured. This is achieved by using a discovery entry in the `clusterd.xml` configuration file. The configuration properies are:

- The JGroups group name
- The JGroups stack configuration file name
- A `boolean` allowing to activate (if `true`) the discovery.

Example:

```
  <discovery>
    <group-name>JGroupsDiscovery</group-name>
    <stack-file>jgroups-discovery.xml</stack-file>
    <start-up>true</start-up>
  </discovery>
```

2.4.6. ear service configuration

The **ear** service allows deployment of complete Java EE applications (including `ejb-jar`, `war` and rar files packed in an `ear` file). This service is based on the **web** service for deploying the included `wars`, the **ejb2** or **ejb3** service for deploying the EJB containers for the included `ejb-jars` and the **resource** service for deploying the included `rars`.

In development mode, as all other Java EE archives ear archives can be deployed automatically as soon as they are copied under `$JONAS_BASE/deploy` (or under another configuration-defined directory) and undeployed as soon as they has been removed from this location.

Here is the part of `jonas.properties` concerning the **ear** service:

```
###################### JOnAS EAR service configuration
#
#  Set the name of the implementation class of the ear service.
jonas.service.ear.class    org.ow2.jonas.ear.internal.JOnASEARService

#  Set the XML deployment descriptors parsing mode for the EAR service
#  (with or without validation).
jonas.service.ear.parsingwithvalidation    true            1
```

```
# Generate stubs for all EJBs that may be accessed from the application
# In almost all cases, this is not required to be enabled as stubs can be found.
jonas.service.ear.genstub    true

# Create a child classloader when deploying EJB3 of the EAR
jonas.service.ear.useEJB3ChildClassloader    true
```

1 Set or not the XML validation at the deployment descriptor parsing time

2.4.7. ejb2 Service configuration

This service provides containers for EJB2.1 components.

An EJB container can be created from an `ejb-jar` file using one of the following possibilities:

- The `ejb-jar` file has been copied under $JONAS_BASE/deploy
- The `ejb-jar` file is packaged inside an ear file as a component of a Java EE application. The container will be created when the Java EE application will be deployed via the **ear** service.
- EJB containers may be dynamically created from ejb-jar files using the JonasAdmin tool.
- EJB containers may be dynamically created from ejb-jar files using the command jonas admin:

```
jonas admin -a <some-dir>/sb.jar
```

The **ejb** service can (and by default does) provide monitoring options. Monitoring provides the following values at a per-EJB basis for stateful and stateless beans:

- **Number of calls** done on all methods.
- **Total business time**, i.e. the time spent executing business (applicative) code.
- **Total time**, i.e. the total time spent executing code (business code + container code).

The **warningThreshold** option can be used to generate a warning each time a method takes more than **warningThreshold** milliseconds to execute. By default, **warningThreshold** is set to 20 seconds.

Here is the part of `jonas.properties` concerning the **ejb2** service:

```
###################### JOnAS EJB 2 Container service configuration
#
#  Set the name of the implementation class of the ejb2 service
jonas.service.ejb2.class    org.ow2.jonas.ejb2.internal.JOnASEJBService

#  Set the XML deployment descriptors parsing mode (with or without validation)
jonas.service.ejb2.parsingwithvalidation    true

# If enabled, the GenIC tool will be called if :
#  - JOnAS version of the ejb-jar is not the same version than the running JOnAS instance
#  - Stubs/Skels stored in the ejb-jar are not the same than the JOnAS current protocols.
# By default, this is enabled
jonas.service.ejb2.auto-genic    true

# Arguments for the auto GenIC (-invokecmd, -verbose, etc.)
jonas.service.ejb2.auto-genic.args -invokecmd

#  Note: these two settings can be overriden by the EJB descriptor.
#
#  If EJB monitoring is enabled, statistics about method call times will be
#  collected. This is a very lightweight measurement and should not have much
#  impact on performance.
jonas.service.ejb2.monitoringEnabled true
#  If EJB monitoring is enabled, this value indicates after how many
#  milliseconds spent executing an EJB method a warning message should be
#  displayed. If 0, no warning will ever be displayed.
jonas.service.ejb2.warningThreshold 20000
```

For customizing the **ejb2** service it is possible to:

- Set or not the XML validation at the deployment descriptor parsing time
- Set or not the automatic generation via the GenIC tool
- Specify the arguments to pass to the GenIC tool

2.4.8. ejb3 service configuration

The **ejb3** service provides EJB 3 container support. This service is provided by the EasyBeans [http://www.easybeans.net] container.

The declaration of the ejb3 service is done in the `jonas.properties` file.

```
###################### JOnAS EJB 3 container service configuration
#
#  Set the name of the implementation class of the EJB 3 service.
jonas.service.ejb3.class    org.ow2.jonas.ejb.easybeans.EasyBeansService

#  List (comma separated) of JPA providers: hibernate,eclipselink
jonas.service.ejb3.jpa.providers hibernate
```

The `jonas.service.ejb3.jpa.providers` property allows to define a list of JPA providers that could be used by EasyBeans [http://www.easybeans.net]. For the moment, Hibernate [http://www.hibernate.org] and EclipseLink [http://www.eclipse.org/eclipselink] JPA providers are available.

EasyBeans [http://www.easybeans.net] has its own configuration file that is located in `JONAS_BASE/conf` folder. The name of the configuration file is `easybeans-jonas.xml`.

By default, EasyBeans [http://www.easybeans.net] is using only services provided by JOnAS. Thus, no additional components are required for starting EasyBeans [http://www.easybeans.net].

```
<?xml version="1.0" encoding="UTF-8"?>
<easybeans xmlns="http://org.ow2.easybeans.server">

    <!-- No infinite loop (managed by JOnAS): wait="false"
         Enable MBeans: mbeans="true"
         Disable the naming: naming="false"
         Use JOnAS JACC provider: jacc="false"
         Using JOnAS monitoring: scanning="false"
         Using JOnAS JMX Connector: connector="false"
         Disable Deployer and J2EEServer MBeans: deployer="false" & j2eeserver="false"
    -->
    <config
        wait="false"
        mbeans="true"
        naming="false"
        jacc="false"
        scanning="false"
        connector="false"
        deployer="false"
        j2eeserver="false" />

    <!-- Define components that will be started at runtime -->
    <components>
        <!-- All components are launched by JOnAS -->

        <!-- RMI component will be used to access some of JNDI properties -->
        <!-- But as there are no protocols, no registry is launched. -->
        <rmi />

        <!-- Start smartclient server with a link to the rmi component-->
        <!--smart-server port="2503" rmi="#rmi" /-->
    </components>
</easybeans>
```

The `<config>` element describes EasyBeans [http://www.easybeans.net] configuration properties that may be different for each application server. The settings provided in this file are the JOnAS settings and they shouldn't be modified in almost any cases.

The `<component>` element defines the EasyBeans [http://www.easybeans.net] components that will be started at the startup. Here EasyBeans [http://www.easybeans.net] is integrated in JOnAS, it will thus use JOnAS services like transaction, security, naming, registry.

The smart client component provides a mechanism for downloading classes missing on the client side, from the server side. This allows to have a very small library on the client side and it downloads classes on demand. When this component is enabled, the listening port can be configured. More documentation on the Smart component can be found in EasyBeans documentation.

2.4.9. ha service configuration

The **ha** (High Availability) service is required in order to replicate stateful session beans (SFSBs).

The **ha** service uses JGroups as a group communication protocol (GCP).

Here is the part of `jonas.properties` related to **ha** service:

```
###################### JOnAS HA service configuration
#
# Set the name of the implementation class of the HA service.
jonas.service.ha.class    org.ow2.jonas.ha.internal.HaServiceImpl

# Set the JGroups configuration file name
jonas.service.ha.jgroups.conf jgroups-ha.xml 1

# Set the JGroups group name
jonas.service.ha.jgroups.groupname jonas-rep 2

# Set the SFSB backup info timeout. The info stored in the backup node is removed when the
 timer expires.
jonas.service.ha.gc.period 600 3

# Set the datasource for the tx table
jonas.service.ha.datasource jdbc_1 4

# Reconnection timeout for JGroups Channel, if it's closed on request.
jonas.service.ha.reconnection.timeout 5000 5
```

1. Set the name of the JGroups configuration file.
2. Set the name of the JGroups group.
3. Set the period of time (in seconds) the system waits before cleaning useless replication information.
4. Set the JNDI name of the datasource corresponding to the database where is located the transaction table used by the replication mechanism.
5. Set the delay to wait for a reconnection.

Note

Refer to the clustering guide [clustering_guide.html#faq.jgroups] for issues related to JGroups.

2.4.10. jaxrpc service configuration

The **jaxrpc** service provides a JAX-RPC 1.1 support for applications using J2EE 1.4 style webservices. J2EE 1.4 style webservices are POJO or Stateless EJB 2.x exposed as webservices using old school deployment descriptors (WEB-INF/webservices.xml or META-INF/webservices.xml)

It is based on the Axis 1.x implementation.

Here is the part of `jonas.properties` concerning the **jaxrpc** service:

```
###################### JOnAS JAX-RPC service configuration
#
#  Set the name of the implementation class of the jaxrpc service.
jonas.service.jaxrpc.class    org.ow2.jonas.ws.axis.AxisService
```

```
#  Set the XML deployment descriptors parsing mode for the jaxrpc service (with or without
 validation).
jonas.service.jaxrpc.parsingwithvalidation    true

# Set the Generator to use with wsgen
jonas.service.jaxrpc.wsgen.generator.factory
 org.ow2.jonas.generators.wsgen.generator.ews.EWSGeneratorFactory

# Set the prefix that will be used to compute URL endpoints for web services
# Example of prefix: http://www.mydomain.com:8888
jonas.service.jaxrpc.url-prefix

# Set automatic WsGen mode on/off
# If set to 'true', WsGen will automatically be applied to all deployed archives (EjbJars,
 Webapps, Applications)
jonas.service.jaxrpc.auto-wsgen.engaged true
```

It is possible to :

- Set or not the XML validation at the deployment descriptor parsing time: property jonas.service.jaxrpc.parsingwithvalidation
- Enforce the URL to be used for the deployed endpoints: property:jonas.service.jaxrpc.url-prefix

  This is interesting when there is a cluster of JOnAS instances and a unique HTTP frontend for load balancing. For example the administrator wants all your web services endpoint to use the `http://www.mydomain.com:8888` URL instead of the usual `http://localhost:9000` (that has a meaning only at local level).
- Enable or not to run the WSGen tool on ejb-jar, war, ear and application client at deployment time.

2.4.11. jaxws service configuration

The **jaxws** service provides a JAX-WS 2.0 support for applications using Java EE 5 style webservices.

It is required to declare it if the configured JOnAS instance have to deploy Java EE application or modules that contains `@WebService` annotated classes (for the service endpoints) or if they contains `@WebServiceRef` / `@WebServiceRefs` (for client side usage of a webservice).

Note

If this service is not activated and deployed applications are using JAX-WS 2.0 APIs, `@WebService` annotated POJO or EJB 3.0 will not be exposed as webservices and `@WebServiceRef` fields/methods will not be injected (may conduct to `NullPointerException`s).

Here is the part of `jonas.properties` concerning the **jaxws** service:

```
###################### JOnAS JAX-WS 2.x service configuration
#
#  Set the name of the implementation class of the jaxws service
jonas.service.jaxws.class    org.ow2.jonas.ws.cxf.CXFService
```

2.4.12. jmx service configuration

The **jmx** service is a mandatory service, so its automatically started in order to administrate or instrument the JOnAS server. It uses the JMX extensions provided by the current Java EE platform.

The **jmx** service creates at stratup, one or more JMX Remote connectors (one for each protocol configured in CAROL, seeSection 2.2, “Configuring the communication protocol and JNDI”). This allows remote management for JMX-based administration applications. A connector's address is based on the corresponding protocol's URL, the protocol name and the server name.

Let's consider the default CAROL configuration, where the RMI/JRMP protocol is used with the following URL:

```
carol.protocols=jrmp
carol.jrmp.url=rmi://localhost:1099
```

The address of the JMX Remote connector for a server named *myJonas* is:

```
service:jmx:rmi:///jndi/rmi://localhost:1099/jrmpconnector_myJonas
```

The **jmx** service can be started in **secured** or **non-secured** mode:

- In non-secured mode, the JOnAS server accepts JMX connections directly, without requiring the client to provide any credentials (no user names or passwords). This implies that any person that has access to the JOnAS server's JMX port (by default, its RMI/JRMP port) can also take any action on the server (including remote code execution).
- In secured mode, any client that connects to the JOnAS server via JMX must provide a valid user name and password.
  - When connecting, the client provides a user name and password by setting the **JMXConnector.CREDENTIALS** key of the properties passed to the connection (**env** variable of the **JMXConnector.connect** method).

    This user name and password is always directly transmitted to the JOnAS server the client is connecting to, and it's that server's decision whether:

    - The user name and password is considered as being valid, therefore the connection will be accepted. This phase is called **Authentication** .
    - That user has the right of accessing a certain method of a certain instance. This phase is called **Authorization**.
  - For authentication, you can use any JAAS LoginModule of the JMX extensions provided by your platform.

    For authorization, you can use any Security Manager provided by your platform.

Here is the part of `jonas.properties` concerning the **jmx** service:

```
###################### JOnAS JMX service configuration
#
#  Set the name of the implementation class of the JMX service
jonas.service.jmx.class                       org.ow2.jonas.jmx.internal.JOnASJMXService

#  Set to true if the JMXRemote interface should require the client to provide
#  authentication information. That information is provided when establishing
#  the JMX connection, using the JMXConnector.CREDENTIALS key.
#
#  Note that if you enable JMX security for a server, all clients (including
#  any administration tool such as the domain master) connecting to this
#  instance via JMX must provide a valid user name and password.
jonas.service.jmx.secured                     false

#  If jonas.service.jmx.secured is set to true, defines the authentication
#  method and the method's parameter. For example, to use file-based
#  authentication using the conf/jmx.passwords file, define:
#      jonas.service.jmx.authentication.method    jmx.remote.x.password.file
#      jonas.service.jmx.authentication.parameter conf/jmx.passwords
#  You are free to use the authentication provider you wish.
jonas.service.jmx.authentication.method       jmx.remote.x.password.file
jonas.service.jmx.authentication.parameter    conf/jmx.passwords
#  You may for example choose to use JAAS LoginModule for authentication.
#  In this case define the used configuration in the JAAS configuration file
#  using the jonas.service.jmx.authentication.parameter:
#      jonas.service.jmx.authentication.method      jmx.remote.x.login.config
#      jonas.service.jmx.authentication.parameter   jaas-jmx

#  If jonas.service.jmx.secured is set to true, defines the authorization
#  method and the method's parameter. For example, to use file-based
#  authorization using the conf/jmx.access file, define:
#      jonas.service.jmx.authorization.method    jmx.remote.x.access.file
```

```
#      jonas.service.jmx.authorization.parameter conf/jmx.access
#  You are free to use the authorization provider you wish.
jonas.service.jmx.authorization.method          jmx.remote.x.access.file
jonas.service.jmx.authorization.parameter       conf/jmx.access
#  You may for example choose to use role-based authorization manager
#  configured using conf/jmx.rolebased.access file. In this case, define:
#      jonas.service.jmx.authorization.method          jmx.remote.x.access.rolebased.file
#      jonas.service.jmx.authorization.parameter       conf/jmx.rolebased.access
```

2.4.13. jtm service configuration

The **jtm** service is used by **ejb2** service in order to provide transaction management for EJB components as defined in the deployment descriptor. The **jtm** service uses a Transaction manager that may be local or may be launched in another JVM (a remote Transaction manager). Typically, when there are several JOnAS servers working together, one jtm service must be considered as the master and the others as slaves. The slaves must be configured as if they were working with a remote Transaction manager.

By default JOTM [http://jotm.objectweb.org/] is the Transaction manager used.

Here is the part of `jonas.properties` concerning the **jtm** service:

```
###################### JOnAS JTM Transaction service configuration
#
#  Set the name of the implementation class of the jtm service
jonas.service.jtm.class    org.ow2.jonas.tm.jotm.JOTMTransactionService

#  Set the Transaction Manager launching mode.
#  If set to 'true', TM is remote: TM must be already launched in an other JVM.
#  If set to 'false', TM is local: TM is going to run into the same JVM
#  than the jonas Server.
jonas.service.jtm.remote    false

#  Set the default transaction timeout, in seconds.
jonas.service.jtm.timeout    60
```

For customizing the **jtm** service It is possible to

- Indicate if the Transaction Manager used in this instance is collocated or remote: jonas.service.jtm.remote property
- Setting the value of the transaction time-out, in seconds: jonas.service.jtm.timeout property

2.4.14. mail service configuration

The **mail** service is an optional service that may be used to send emails.

It is based on JavaMail and on JavaBeans Activation Framework (JAF) API.The default implementation of the **mail** service rely on the GNUMail implementation of these API.

A mail factory is required in order to send or receive mails. JOnAS provides two types of mail factories: `javax.mail.Session` and `javax.mail.internet.MimePartDataSource`. `MimePartDataSource` factories allow mail to be sent with a subject and the recipients already set.

Mail factory objects must be configured according to their type. The subsections that follow briefly describe how to configure `Session` and `MimePartDataSource` mail factory objects, in order to run the SessionMailer SessionBean and the MimePartDSMailer SessionBean delivered with the platform.

Here is the part of `jonas.properties` concerning the **mail** service:

```
###################### JOnAS Mail service configuration
#
#  Set the name of the implementation class of the mail service
```

```
jonas.service.mail.class org.ow2.jonas.mail.internal.JOnASMailService

#  Set the jonas mail factories.
#  This property is set with a coma-separated list of MailFactory properties
#  file names (without the '.properties' suffix).
#  Ex: MailSession1,MailMimePartDS1 (while the properties file names are
#                  MailSession1.properties and MailMimePartDS1.properties)
jonas.service.mail.factories
```

Mail factory objects created by JOnAS must be given a name. For example, consider two factories called `MailSession1` and `MailMimePartDS1`. Each factory must have a configuration file whose name is the name of the factory with the `.properties` extension (`MailSession1.properties` for the MailSession1 factory).

For this example jonas.service.mail.factories property must be set to:

```
jonas.service.mail.factories MailSession1,MailMimePartDS1
```

2.4.14.1. Configuring Session mail factory

A template `MailSession1.properties` file is supplied in `$JONAS_BASE/conf.` It contains two mandatory properties :

```
#Factory Name/Type
mail.factory.name mailSession_1
mail.factory.type javax.mail.Session
```

The JNDI name of the mail factory object is mailSession_1. This template must be updated with values appropriate to your installation. See the section "Configuring a mail factory" below for the list of available properties.

2.4.14.2. Configuring MimePartDataSource mail factory

A template `MimePartDS1.properties` is supplied in `$JONAS_BASE/conf`. It contains two mandatory properties :

```
#Factory Name/Type
mail.factory.name mailMimePartDS_1
mail.factory.type javax.mail.internet.MimePartDataSource
```

The JNDI name of the mail factory object is mailMimePartDS_1. This template must be updated with values appropriate to your installation. See the section "Configuring a mail factory" below for the list of available properties.

2.4.14.3. Configuring a mail factory

Here are the possible properties

- Required properties:

Property name	Description
mail.factory.name	JNDI name of the mail factory
mail.factory.type	The type of the factory. This property can be javax.mail.Session or javax.mail.internet.MimePartDataSource.

- Optional properties: Authentication properties

Property name	Description
mail.authentication.username	Set the username for the authentication.

mail.authentication.password	Set the password for the authentication.

- Optional properties: javax.mail.Session.properties (refer to JavaMail documentation for more information)

Property name	Description
mail.debug	The initial debug mode. Default is false.
mail.from	The return email address of the current user, used by the InternetAddress method getLocalAddress.
mail.mime.address.strict	The MimeMessage class uses the InternetAddress method parseHeader to parse headers in messages. This property controls the strict flag passed to the parseHeader method. The default is true.
mail.host	The default host name of the mail server for both Stores and Transports. Used if the mail.protocol.host property is not set.
mail.store.protocol	Specifies the default message access protocol. The Session method getStore() returns a Store object that implements this protocol. By default the first Store provider in the configuration files is returned.
mail.transport.protocol	Specifies the default message access protocol. The Session method getTransport() returns a Transport object that implements this protocol. By default, the first Transport provider in the configuration files is returned.
mail.user	The default user name to use when connecting to the mail server. Used if the mail.protocol.user property is not set.
mail.<protocol>.class	Specifies the fully- qualified class name of the provider for the specified protocol. Used in cases where more than one provider for a given protocol exists; this property can be used to specify which provider to use by default. The provider must still be listed in a configuration file.
mail.<protocol>.host	The host name of the mail server for the specified protocol. Overrides the mail.host property.
mail.<protocol>.port	The port number of the mail server for the specified protocol. If not specified, the protocol's default port number is used.
mail.<protocol>.user	The user name to use when connecting to mail servers using the specified protocol. Overrides the mail.user property.

- Optional properties:MimePartDataSource properties (Only used if mail.factory.type is javax.mail.internet.MimePartDataSource)

Property name	Description
mail.to	Set the list of primary recipients ("to") of the message.
mail.cc	Set the list of Carbon Copy recipients ("cc") of the message. mail.bcc

mail.bcc	Set the list of Blind Carbon Copy recipients ("bcc") of the message.
mail.subject	Set the subject of the message.

2.4.15. registry service configuration

This service is used for accessing the RMI registry, CMI registry, or the CosNaming (RMI/IIOP), depending on the configuration of communication protocols specified in `carol.properties`, refer to Section 2.2, “Configuring the communication protocol and JNDI” .

Here is the part of `jonas.properties` file concerning the **registry** service.

```
###################### JOnAS Registry service configuration
#
#  Set the name of the implementation class of the Registry service
jonas.service.registry.class org.ow2.jonas.registry.carol.CarolRegistryService

#  Set the Registry launching mode
#  If set to 'automatic', the registry is launched in the same JVM as Application Server,
#                         if it's not already started.
#  If set to 'collocated', the registry is launched in the same JVM as Application Server
#  If set to 'remote', the registry has to be launched before in a separate JVM
jonas.service.registry.mode    collocated
```

2.4.16. resource service configuration

The **resource** service must be started when Java EE components require access to an external Enterprise Information Systems. The standard way to do this is to use a third party software component called **Resource Adapter**.

The role of the Resource service is to deploy the Resource Adapters in the JOnAS server, i.e., configure it in the operational environment and register in JNDI name space a connection factory instance that can be looked up by the application components. The **resource** service implements the Java EE Connector Architecture 1.5[6].

Resource Adapter are packaged in Java EE rar archives.

In development mode, as all other Java EE archives rar archives can be deployed automatically as soon as they are copied under $JONAS_BASE/deploy and undeployed as soon as they has been removed from this location.

For more information see Section 2.4.3, “depmonitor service configuration”.

The other ways to deploy rar archives is

- to use the jonasAdmin console.
- to use the command jonas admin:

```
jonas admin -a   <mydir>/myrar.rar
```

A JOnAS specific resource adapter configuration xml file must be included in each resource adapter. This file replicates the values of all configuration properties declared in the deployment descriptor for the resource adapter. Refer to Defining the JOnAS Connector Deployment Descriptor in J2EE Connector Programmer's Guide [connector_pg.html] for additional information.

Here is the part of `jonas.properties` related to **resource** service:

[6]There is no real acronym for this specification JCA was the acronym for Java Cryptography Architecture . In the rest of this document we will use J2CA

```
###################### JOnAS J2CA resource service configuration
#
#  Set the name of the implementation class of the J2CA resource service
jonas.service.resource.class org.ow2.jonas.resource.internal.JOnASResourceService
```

The worker thread pool used for all J2CA 1.5 Resource Adapters deployed can be configured in the Section 2.4.22, “wm service configuration” service.

resource service is mainly used in JOnAS for accessing databases via a JDBC resource adapter (in this case it replace **dbm** service) and for providing JMS facilities.

JOnAS provides several JDBC resource adapters and a JMS resource adapter on top of JORAM [http://joram.objectweb.org/] More information about configuring resource adapters can be found in Section 2.6, “Configuring JDBC Resource Adapters”

2.4.17. security service configuration

Here is the part of `jonas.properties` related to **security** service:

```
#
###################### JOnAS SECURITY service configuration
#
#  Set the name of the implementation class of the security service
jonas.service.security.class  org.ow2.jonas.security.internal.JonasSecurityServiceImpl

# Realm used for CsiV2 authentication
jonas.service.security.csiv2.realm    memrlm_1

# Realm used for Web Service authentication
jonas.service.security.ws.realm    memrlm_1

# Registration of realm resources into JNDI
# Disable by default so configuration is not available with clients
jonas.service.security.realm.jndi.registration    false

# Enable security context check on Remote Login Module
jonas.security.context.check    false

# Path to the keystore file
jonas.security.context.check.keystoreFile /tmp/keystore

# Pass used for the keystore file
jonas.security.context.check.keystorePass keystorepass

# Alias (stored in the keystore)
jonas.security.context.check.alias FB
```

In fact properties jonas.service.security.csiv2.realm and jonas.service.security.ws.realm are only useful for users that use security on top of rmi/iiop or on top of web services . in these case with `memrlm_1` it is possible to make a link to the memomyrealm named `memrlm_1` in the $JONAS_BASE/conf/jonas-realm.xml file and retrieve users name and roles.

Don't forget that for using security the global property jonas.security.propagation to true and that an important property related to security is jonas.security.manager see Section 2.1.5.1, “Global properties”

All other security configuration related to JOnAS is done in the file `jonas-realm.xml` and security configuration related to web containers, certificate, etc., is done in the appropriate files. Refer to the subsection Section 2.5, “Configuring Security” for a complete description of security configuration.

2.4.18. smartclient service configuration

The **smartclient** lets remote clients download classes and other resources necessary for connecting to JOnAS services (JNDI context factories, EJB3 interceptors, ...) directly from the JOnAS server they're dealing with.

This way, the heavy clients only need to include a lightweight JAR file for the Smartclient client and are always guaranteed to have the good versions of all components.

Here is the part of `jonas.properties` concerning the **smartclient** service:

```
###################### JOnAS/EasyBeans Smartclient service configuration
#
#  Set the name of the implementation class of the smartclient service.
jonas.service.smartclient.class
 org.ow2.jonas.smartclient.internal.SmartclientServiceImpl
# port number the Smartclient service listens on
jonas.service.smartclient.port          2503
```

2.4.19. versioning service configuration

2.4.19.1. About the versioning service

This service has been designed for dynamic redeployment of applications, without any application downtime and without users' sessions being lost:

- Deployment of a new version of an application does not require the undeployment of any previous version.
- Users that were on a previous version keep on using that version as long as their session on that version is active (for example, as long as they have not finished buying items on the previous version of the online trade web site). This guarantees that no user data will be lost, and that if there is any problem with the new version the old version is still available instantly.
- New versions of the application can be deployed using various strategies, for instance allow testing of the new version by a small community to ensure its readiness for production.

The versioning service achieves this by adding virtual contexts to all services that provide support for versioning. To use the versioning service:

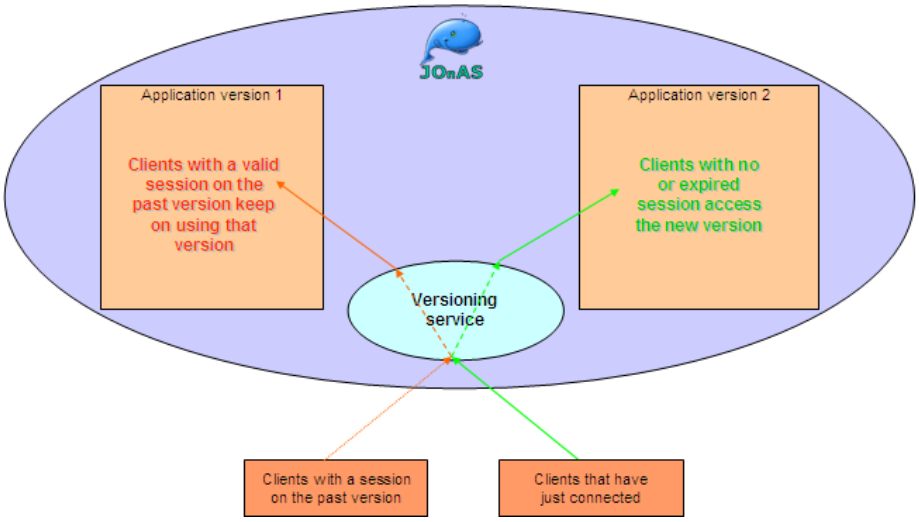
1. Enable the versioning service in `jonas.properties`
2. Define the **Implementation-Version** attribute in your deployable file's (whether `war`, `jar` or `ear`) `MANIFEST`. Note that:
   - ANT, Maven as well as most IDEs can set any `MANIFEST` attribute automatically.
   - If the archive that will be deployed is an `ear`, the **Implementation-Version** defined in the `MANIFEST` of the ear will be used for all archives the `ear` contains.

When the versioning service is enabled, application resources (web pages, EJBs, etc.) are accessed the following way:

- Each versioned application has a user (virtual) address. Each version of an application is renamed and bound to that virtual address. Each bound version has a policy (that can be changed in time in order to manage the deployment strategy):
  - **Private**: Can only be accessed by clients that satisfy some prerequisites; for example belong to a certain IP address group or provide a certain credential.
  - **Reserved**: Not accessible using the virtual address, therefore can only be accessed directly (using the versioned address).
  - **Disabled**: Only accessible by clients that have been using this version in the past (until their session expires). This guarantees that users will not lose their session data during a redeployment.
  - **Default**: Version accessed by all clients that don't fit in any other policy.

- A user can access the application resource indirectly (using the virtual address) or directly (using the versioned address).
- If the user tries to access the application resource indirectly (using the virtual address), the versioning system:
  - First checks if that user is defined as a user that can access a version of the application with the **Private** access policy. If that is the case, the user is routed to that private version of the application.
  - Then checks if that user already has a session in a version of the application with the **Disabled** access policy. If that is the case, the user is routed to that disabled version of the application.
  - If neither of these cases are true, routes the user to the version of the application with the **Default** access policy. If the application does not define any default version, the user will see "*resource not found*" message.

This can be schematized as follows:



The current limitations of the versioning service are:

- Only the Tomcat Web Container supports the versioning service. That support is fully functional, recognition of users is based on the session ID (via cookie or GET).
- Both EJB2 and EJB3s support the versioning service. That support is fully functional, EJB lookups in the same EAR always redirect to the same version.
- Web Service support for the versioning service is in design phase.
- The Private context policy has not been implemented.

As this service allows seamless and interruptionless upgrade and test of all applications, it is strongly recommended for all applications to refer version identifiers in their manifest files. Remember that ANT, Maven as well as most IDEs can set any `MANIFEST` attribute automatically.

We will now detail the way versioning works by creating two versions of the **Java EE 5 Sample Application** in the JOnAS `examples` folder: version 1.0.0 and version 1.0.1. Since the application is an EAR, we only need to refer the version identifier in the EAR file.

2.4.19.2. Focus: versioned Web Applications

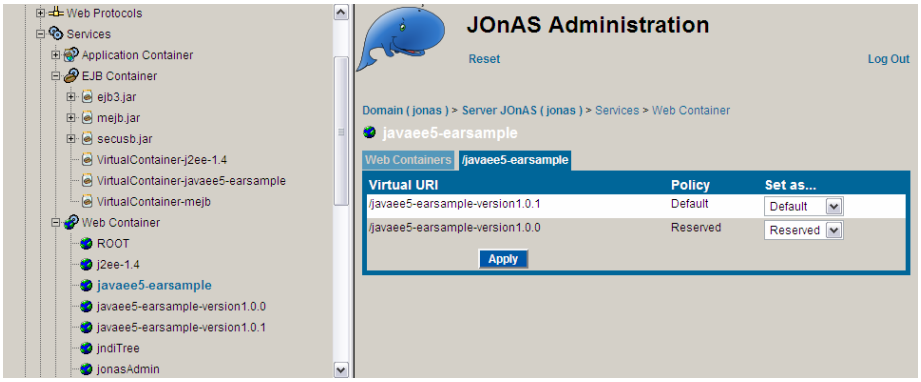
When the first version of the EAR is deployed:

- The application gets deployed on the URI **/javaee5-earsample-version1.0.0**.

- The virtual URI **/javaee5-earsample** is created.
- The real URI **/javaee5-earsample-version1.0.0** gets bound to the virtual URI **/javaee5-earsample**.

Therefore, when a user accesses the **/javaee5-earsample** URI, the content seen is the same as the one on **/javaee5-earsample-version1.0.0**.

We now deploy the second version of the application, version **1.0.1**, via the JOnAS Web Admin panel. When the second version is deployed, it is bound to the virtual URI as **Reserved** (this is the default policy and can be modified via JMX at any time). This means that the only way of accessing the **1.0.1** version of the application is to type as URI **/javaee5-earsample-version1.0.1**. All visitors of **/javaee5-earsample** will still access the version **1.0.0**.



To change the access policies of each version of the virtual URI (the only URI end users are expected to access), go to the list of **Web Container**s in the JOnAS Web Admin panel. If you set the version **1.0.0** as **Disabled** and the version **1.0.1** as **Default**:

- The user that was on **/javaee5-earsample** when the default version was **1.0.0** will stay on version **1.0.0** until her/his session expires (i.e. the browser window is closed).
- Any user that connects to **/javaee5-earsample** for the first time will visit version **1.0.1**.

2.4.19.3. Focus: versioned EJBs

When the first version of the EAR is deployed:

- All EJBs that register on the JNDI directory register with a prefixed name, which is their original name prefixed by **javaee5_earsample_version1.0.0/**. For example, the **myStateless** bean gets registered as **javaee5_earsample_version1.0.0/myStateless**.
- For each EJB, the original JNDI name is also registered and points exactly to the same JNDI link.

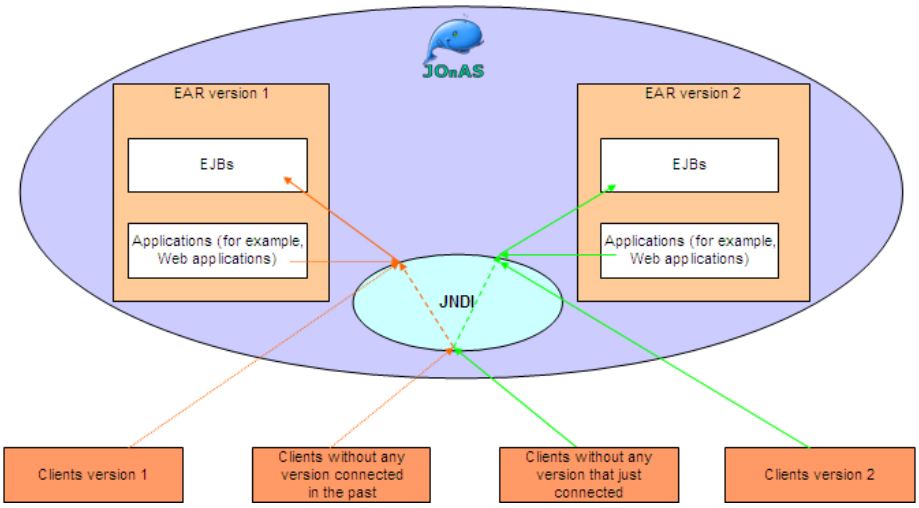
Therefore, when a user looks up for the **myStateless** bean, the reference received is the same as the one received when **javaee5_earsample_version1.0.0/myStateless** is looked up.

The behaviours of the Web and EJB services when the version **1.0.1** is deployed are similar, except for one very important point: when multiple applications are packaged together, the only versions of the applications they've been tested against are the versions inside the same EAR. Therefore, blindly accessing the **Default** version of the EJBs could have unexpected results. This is why the concept of versioned EJB clients has been created:

- All EJB clients in a versioned EAR automatically become versioned EJB clients. Their target version is the version of the host EAR, which implies that intra-EAR accesses are always done to the same version.
- External EJB clients can also specify the versions for the EJB they need to access.

- Non-versioned external EJB clients will access the **Private**, **Reserved**, **Disabled** or **Default** versions as usual.

This can be schematized as follows:



As with the versioned Web Applications, to change the access policies of each version of the virtual JNDI container (which knows the JNDI names end users are expected to access), go to the list of **EJB Container**s in the JOnAS Web Admin panel. If you set the version **1.0.0** as **Disabled** and the version **1.0.1** as **Default**:

- All clients that know about the versioned JNDI names (remember that this will always be the case in a versioned EAR application) will always access the version they specify.
- References to **myStateless** obtained before this operation will stay on version **1.0.0**.
- Any user that looks up **myStateless** for the first time will get a reference to version **1.0.1**.

2.4.20. wc service configuration

The **wc** service allows to clean up periodically the work directory of the JOnAS server. This service don't need to be defined in the list of JOnAS services as it is automatically started when JOnAS is in development mode.

During the deployment process of an application, a specific working directory is created and associated to the application. After a defined time, the clean task tries to delete working directories of applications which have been undeployed. Note that the clean task is already executed at startup of the JOnAS server.

Here is the part of `jonas.properties` concerning the **wc** service:

```
###################### JOnAS WorkCleaner service configuration
#
#  Set the name of the implementation class of the wc service
jonas.service.wc.class    org.ow2.jonas.workcleaner.internal.JOnASWorkCleanerService

#  Set the clean period in seconds
jonas.service.wc.period   300                                    1
```

1 Define the period between two executions of the clean task (in seconds)

2.4.21. web service configuration

This service provides containers for the web components used by the Java EE applications.

JOnAS provides two implementations of this service: one for Jetty 6.x, one for Tomcat 6.x. It is necessary to run this service in order to use the JonasAdmin tool. A web container is created from a `war` file.

In development mode, as all other Java EE archives war archives can be deployed automatically as soon as they are copied under $JONAS_BASE/deploy and undeployed as soon as they has been removed from this location.

Here is the part of `jonas.properties` concerning the **web** service:

```
#
###################### JOnAS Web container service configuration
#
#  Set the name of the implementation class of the web container service.
jonas.service.web.class    org.ow2.jonas.web.tomcat6.Tomcat6Service
#jonas.service.web.class    org.ow2.jonas.web.jetty6.Jetty6Service

#  Set the XML deployment descriptors parsing mode for the WEB container
#  service (with or without validation).
jonas.service.web.parsingwithvalidation    true                          1

#  If true, the onDemand feature is enabled. A proxy is listening on the http port and will
 make actions like starting or deploying applications.
#  The web container instance is started on another port number (that can be specified) but
 all access are proxified.
#  It means that the web container will be started only when a connection is done on the
 http port.
#  The .war file is also loaded upon request.
#  This feature cannot be enabled in production mode.
jonas.service.web.ondemand.enabled    true                               2

#  The redirect port number is used to specify the port number of the http web container.
#  The proxy will listen on the http web container port and redirect all requests on this
 redirect port
#  0 means that a random port is used.
jonas.service.web.ondemand.redirectPort    0                             3
```

For customizing the **web** service, it is possible to:

1. Set or not the XML validation at the deployment descriptor parsing time.
2. Enable or not the onDemand feature. In addition of activating this global feature, each web application that has to be loaded on demand must declare the *on-demand* element in the JOnAS deployment descriptor (`WEB-INF/jonas-web.xml`) as below:

   ```
   <?xml version="1.0" encoding="ISO-8859-1"?>
   <jonas-web-app 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-web-app_5_1.xsd">

     ...

     <!-- Load this application on demand (if enabled in the webcontainer service) -->
     <on-demand>true</on-demand>
   </jonas-web-app>
   ```

3. This property is specific to the onDemand feature. Useful to set the port number of the http web container in case of the port number defined in the web server configuration is used by the proxy.

2.4.22. wm service configuration

The **wm** service provides a J2CA WorkManager [http://java.sun.com/j2ee/1.4/docs/api/javax/resource/spi/work/WorkManager.html] implementation. This service don't need to be defined in the list of JOnAS services as it is automatically started when required.

Here is the part of `jonas.properties` concerning the **wm** service:

```
###################### JOnAS WorkManager service configuration
#
#  Set the name of the implementation class of the wm service
jonas.service.wm.class    org.ow2.jonas.workmanager.internal.JOnASWorkManagerService
```

```
#  Set the size of the worker thread pool
jonas.service.wm.minworkthreads   3                                  1

#  Set the maximun size of the worker thread pool
jonas.service.wm.maxworkthreads   80                                 2

#  Set the max # of seconds that a thread will wait for work
#  This is used to shrink the worker thread pool back to minimum
jonas.service.wm.threadwaittimeout 60                                3
```

1 Defines the minimum size of the Thread pool
2 Defines the maximum size of the Thread pool
3 Defines the maximum time (in seconds) that a worker Thread should wait before execution

The **wm** service is used, for example, in the **resource** service (J2CA 1.5 implementation) in order to provide a `javax.resource.spi.work.WorkManager` instance for deployed resource adapters (like `JMS`, ...).

2.4.23. wsdl-publisher service configuration

The **wsdl-publisher** service provides a pluggable component dedicated to alternate WSDL publishing mechanisms.

By default, all the web services deployed by JOnAS have their WSDL available at a given URL location. For J2EE 1.4 webservices, the URL ends with `?JWSDL`, for Java EE 5.0 webservices, the URL ends with `?WSDL`.

When this default publishing mechansim is not sufficent, it is possible to add one or more custom WSDL publishers. Within JOnAS, 2 custom publishers are availables (file based and JAXR based).

Here is the part of `jonas.properties` concerning the **wsdl-publisher** service:

```
###################### JOnAS WSDL Publisher service configuration
#
#  Set the name of the implementation class of the WSDL Publisher service.
jonas.service.wsdl-publisher.class
 org.ow2.jonas.ws.publish.internal.manager.DefaultWSDLPublisherManager

#  Set the WSDL Publishers list for WSDL publication
#  A minimum of 1 WSDLPublisher is required !
#  This property is set with a coma-separated list of WSDLPublisher properties
#  file names (without the '.properties' suffix).
#  Ex: file1,uddi (while the properties file names are
#                  file1.properties and uddi.properties)
jonas.service.wsdl-publisher.publishers    file1
```

2.4.23.1. File WSDLPublisher

The *File* WSDLPublisher type is used in simple WebServices usage scenario, when the application doesn't requires a full blown web services registry (like UDDI [http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration] or ebXML [http://www.ebxml.org]). It will simply save the WSDL documents (and their dependencies) in a configurable directory.

```
# FileWSDLPublisher class
jonas.service.wsdl.class    org.ow2.jonas.ws.publish.internal.file.FileWSDLPublisher

# Directory where WSDLs will be copied
# If not set JONAS_BASE/wsdls will be used
# jonas.service.publish.file.directory    /tmp     1

# Encoding of the file (In respect with the platform JOnAS is running on)
# If not set default to UTF-8
jonas.service.publish.file.encoding     UTF-8      2
```

1 Base directory where WSDL documents will be published

2 File encoding to be used (must be supported by the platform)

2.4.23.2. JAXR WSDLPublisher

The *JAXR* WSDLPublisher type is responsible of publishing a given WSDL in an enterprise level registry or repository, allowing external clients to get the technical and administrative information about the deployed service.[7]

```
# RegistryWSDLPublisher class
jonas.service.wsdl.class    org.ow2.jonas.ws.publish.internal.registry.RegistryWSDLPublisher

# User name and Password to access Registry
jonas.service.publish.uddi.username                jonas
            1
jonas.service.publish.uddi.password                jonas
            2

# Organization name, small desc (optionnal) and primary contact name.
jonas.service.publish.uddi.organization.name       OW2
            3
jonas.service.publish.uddi.organization.desc       OW2 Consortium (http://www.ow2.org)
            4
jonas.service.publish.uddi.organization.person_name JOnAS
            5

# URLs where Registry can be contacted (Publish an Query APIs)
javax.xml.registry.lifeCycleManagerURL  http://localhost:9000/juddi/publish
            6
javax.xml.registry.queryManagerURL      http://localhost:9000/juddi/inquiry
            7
```

1 Username
2 Credential to be used for registry authentication
3 Organization name
4 Organization details/description
5 Contact for the organization
6 JAXR LifeCycleManager URL (Administration URL where WSDL can be published)
7 JAXR QueryManager URL (Read only URL acting as a registry)

2.5. Configuring Security

The **security** service is used by the **ejb**, **web**, **ws** services to provide security for Java EE components. The **ejb** service provides security in two forms: declarative security and programmatic security that is described in the EJB Programmer's Guide: Security Management [ejb2_programmer_guide.html#ejb2.security] .

The **security** service exploits security roles and method permissions located in the Java EE deployment descriptors.

A main concept in security is *Authentication* which is the mechanism telling the container the identity of the user making the current request.

A caller is a client that may be a servlet client or a container client. Usually a client proves its identity by a couple user/password or a certificate (*credential*). Once the identification is correct JOnAS must build a security context that will be propagated with requests and be used by the container to verify that the user exists and has permissions sufficient to make the request.

JAAS is a standard framework for authenticating users. It defines configuration files (`jaas.config`) and interfaces like the `LoginModule` interface that may be used in JOnAS to perform authentication tasks.

Lightweight authentication mechanism using JACC may be used to authenticate servlet client.

[7] JOnAS has been tested with Apache jUDDI, an ASL2 UDDI v2 implementation.

In the Tomcat documentation we can find this definition: “A Realm is a "database" of usernames and passwords that identify valid users of a web application (or set of web applications), plus an enum eration of the list of roles associated with each valid user.”

In both authentication mechanisms the container use a *realm* to verify validity of users. In JOnAS the *realm* may be a database accessed via JDBC (Database realm), a LDAP directory (LDAP realm) or a flat file (Memory realm). The type of realm to use is specified in `$JONAS_BASE/conf/jonas-realm.xml`.

2.5.1. jonas-realm.xml

The file `$JONAS_BASE/conf/jonas-realm.xml` file describes:

- the content of flat file memory realm
- how to access a database realm
- how to access a LDAP realm

2.5.1.1. Memory realm

The *memoryrealm* must be named and defines users, groups and roles in the section <jonas-memoryrealm>

```
<jonas-memoryrealm>
 <memoryrealm name="memrlm_1"> 1
   <roles>
     <role name="jonas-admin" description="JonasAdmin role" /> 2
     <role name="tomcat" description="Used in examples" />
   </roles>
   <groups>
   <group name="jonas"
      roles="jonas-admin,tomcat,jaas,ws-security" description="All authorization" /> 3
   </groups>
   <users>
     <user name="tomcat" password="tomcat" roles="tomcat,jonas-admin,manager" /> 4
     <user name="jetty" password="jetty" roles="jetty" />
     <!-- Example of a crypt password : password for jadmin is : jonas -->
     <user name="jadmin" password="{MD5}nF3dVBB3NPfRgzWlJFwoaw==" roles="jonas-admin" /> 5
     <user name="jps_admin" password="admin" roles="administrator" />
     <user name="supplier" password="supplier" roles="administrator" />
     <!-- Another crypt example in another format : password is jonas -->
     <!-- JonasAdmin uses name="jonas" password="jonas" -->
     <user name="jonas" password="SHA:NaLG+uYfgHeqth+qQBlyKr8FCTw=" groups="jonas" /> 6
     <user name="principal1" password="password1" roles="role1" />
     <user name="principal2" password="password2" roles="role2" />
    </users>
 </memoryrealm>
</jonas-memoryrealm>
```

1 memoryrealm must be named. This name will be used in the web container configuration file
2 definition of a security role
3 definition of a group of roles
4 definition of a user with non encrypted password and a list of roles
5 definition of a user with encrypted password (format MD5)
6 definition of a user with encrypted password (format SHA)

2.5.1.2. database realm

Users, groups, and roles information are stored in a database; the configuration for accessing the corresponding database is described in the section <jonas-dsrealm>

The configuration requires the name of a datasource, the tables used, and the names of the columns.

```
<jonas-dsrealm>
```

```
   <dsrealm name="dsrlm_1" 1
            dsName="jdbc_1" 2
      userTable="realm_users" userTableUsernameCol="user_name"
 userTablePasswordCol="user_pass" 3
      roleTable="realm_roles" roleTableUsernameCol="user_name"
 roleTableRolenameCol="role_name"> 4
   </dsrealm>
</jonas-dsrealm>
```

1 dsrealm must be named
2 JNDI name of the `dataSource` for accessing the database via JDBC
3 defines the name of the user table and the name of the columns for username/password
4 defines the name of the role table and the name of the columns for username/rolename

to use this database a `Datasource` configuration with the right JNDI name for the **dbm** service must be set in the `jonas.properties` file.

2.5.1.3. LDAP realm

Users, groups, and roles information are stored in an LDAP directory. This is described in the section <jonas-ldaprealm>

There are some optional parameters. If they are not specified, some of the parameters are set to a default value. For example if the providerUrl element is not set, the default value is `ldap://localhost:389`. The jonas-realm_1_0.dtd DTD [http://jonas.objectweb.org/dtds/jonas-realm_1_0.dtd]file show the default values.

- minimal example:

```
<jonas-ldaprealm>
<ldaprealm name="ldaprlm_1" 1
     baseDN="dc=jonas,dc=ow2,dc=org" /> 2
</jonas-ldaprealm>
```

1 ldaprealm must be named
2 to access to LDAP server

For this sample, it is assumed that the LDAP server is on the same computer and is on the default port (389).

2.5.2. Servlet Authentication

Depending on the servlet container used, configuration differs.

2.5.2.1. Authentication with User/password and Tomcat 6

- Tomcat configuration:

  Tomcat embedded in the JOnAS distribution is configured in `$JONAS_BASE/conf/tomcat6-server.xml` to use the memory realm named `memrlm_1`

```
<Server>
[...]
<Realm className="org.ow2.jonas.web.tomcat6.security.Realm" resourceName="memrlm_1" />
[...]
</Server>
```

  The authentication mechanism implemented by the class `org.ow2.jonas.web.tomcat6.security.Realm` is able to work with database or LDAP realm configured in `jonas-realm.xml`. The value of resourceName attribute identifies the *realm* to be used in `jonas-realm.xml`.

- webapp configuration:

In the `web.xml` of the web application a *basic authentication* or a *Form based authentication* may be used

```
<web-app>
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Example Basic Authentication Area</realm-name>
  </login-config>
</web-app>
```

or

```
<web-app>
 <login-config>
   <auth-method>FORM</auth-method>
   <form-login-config>
   <form-login-page>login.jsp</form-login-page>
   <form-error-page>error.jsp</form-error-page>
   </form-login-config>
   </login-config>
</web-app>
```

Like basic authentication, form-based authentication is not secure, since the content of the user dialog is sent as plain text, and the target server is not authenticated.

To overcome this vulnerability the authentication protocol may be run over a SSL session that ensures that all message contents are protected for confidentiality.

2.5.2.2. Authentication with certificate and Tomcat 6

In this case, users will not have to enter a login/password. They will just present their certificates and authentication is performed transparently by the browser (after the user has imported his certificate into it). Therefore, the identity presented to the server is not a login, but a Distinguished Name(DN).

- jonas-realm configuration:

  The name identifying the person to whom the certificate belongs looks like the following: CN=Someone Unknown, OU=ObjectWeb, O=JOnAS, C=ORG with:

  CN : Common Name

  OU : Organizational Unit

  O : Organization

  C : Country Name

  E : Email Address

  L : Locality

  ST :State or Province Name

  The Subject in a certificate contains the main attributes and may include additional ones, such as Title, Street Address, Postal Code, Phone Number.

  In the jonas-realm.xml a user with password looks like:

  ```
  <user name="jps_admin" password="admin" roles="administrator"/>
  ```

  A certificate-based user must have its DN preceded by the String: ##DN## example:

  ```
  <user name="##DN##CN=whale, OU=ObjectWeb, O=JOnAS, L=JOnAS, ST=JOnAS, C=ORG"
        password="" roles="jadmin" />
  ```

- Tomcat Realm configuration:

  The current Realm in $JONAS_BASE/conf/tomcat6-server.xml must be replaced by:

```
<Server>
[...]
<Realm className="org.ow2.jonas.web.tomcat6.security.Realm" />
[...]
</Server>
```

  The class specified uses the JAAS model to authenticate the users. Thus, to choose the correct realm to be used for authentication, JAAS must be configured see in Section 2.5.4, “JAAS configuration”.

- Tomcat SSL configuration:

  The following example of <connector> element must be uncommented in $JONAS_BASE/conf/tomcat6-server.xml and customized (if necessary):

```
<Server>
[...]
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 9043 -->
    <!--
    <Connector port="9043" maxHttpHeaderSize="8192"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" disableUploadTimeout="true"
               acceptCount="100" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS" />
    -->
[...]
</Server>
```

  A complete description of SSL configuration can be found in SSL Configuration HOW-TO [http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html]

- Webapp configuration:

  In the `web.xml`of the web application a *Client Certificate Authentication Configuration* must be set, a security-constraint may be used if needed; example:

```
<web-app>
   <login-config>
        <auth-method>CLIENT-CERT</auth-method>
        <realm-name>Example Authentication Area</realm-name>
   </login-config>

   <security-constraint>
   ..
     <user-data-constraint>
     <transport-guarantee>CONFIDENTIAL</transport-guarantee>
     </user-data-constraint>
   </security-constraint>
</web-app>
```

2.5.2.3. Servlet Authentication with User/password and Jetty 6.x

- Jetty configuration

  A web-jetty.xml file must be provided in the `WEB-INF` directory in the `.war` file in which a security interceptor `org.ow2.jonas.web.jetty6.security.Realm` form is specified instead of the default one:

```
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
  <Call name="setRealmName">
    <Arg>Example Basic Authentication Area</Arg>
  </Call>
  <Call name="setRealm">
    <Arg>
      <New class="org.ow2.jonas.web.jetty6.security.Realm">
        <Arg>Example Basic Authentication Area</Arg>
        <Arg>memrlm_1</Arg>
```

```
      </New>
    </Arg>
  </Call>
</Configure>
```

- webapp configuration:

  is similar to the webapp configuration with Tomcat see ??? [43].

2.5.2.4. Authentication with certificate and Jetty 6.x

- Jetty Realm configuration:

  Edit the web-jetty.xml file under WEB-INF directory in the .war file to declare a Realm name and a Realm:

```
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
...
!-- Set the same realm name as the one specified in <realm-name> in <login-config>
    in the web.xml file of your web application -->
<Call name="setRealmName">
  <Arg>Example Authentication Area</Arg>
</Call>
<!-- Set the class Jetty has to use to authenticate the user and a title name for
     the pop-up window -->
<Call name="setRealm">
  <Arg>
    <New class="org.ow2.jonas.web.jetty6.security.Realm">
      <Arg>JAAS on Jetty</Arg>
    </New>
  </Arg>
</Call>
...
</Configure>
```

  The class specified uses the JAAS model to authenticate the users. Thus, to choose the correct *realm* to be used for authentication, JAAS must be configured, see in Section 2.5.4, “JAAS configuration”.

- Jetty SSL configuration:

  In the global deployment descriptor of Jetty (the jetty6.xml file), located in the $JONAS_BASE/conf directory, uncomment this part:

```
 <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
  <!-- Add a HTTPS SSL listener on port 9043                                  -->
  <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
  <!-- UNCOMMENT TO ACTIVATE
  <Call name="addListener">
    <Arg>
      <New class="org.mortbay.http.SunJsseListener">
        <Set name="Port">9043</Set>
        <Set name="MinThreads">5</Set>
        <Set name="MaxThreads">100</Set>
        <Set name="MaxIdleTimeMs">30000</Set>
        <Set name="LowResourcePersistTimeMs">2000</Set>
        <Set name="Keystore"><SystemProperty name="jetty.home" default="."/>/etc/
demokeystore</Set>
  <Set name="Password">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
  <Set name="KeyPassword">OBF:1u2u1wml1z7s1z7a1wnl1u2g</Set>
      </New>
    </Arg>
  </Call>
  -->
```

  A complete description of howto configure SSL for Jetty may be founf here [http://jetty.mortbay.org/jetty5/faq/faq_s_400-Security_t_ssl.html]

- webapp configuration

  is similar to the webapp configuration with Tomcat [45]

- jonas-realm configuration

  is similar to the jonas-realm configuration with Tomcat [44]

2.5.3. Client container Authentication

To enable authentication mechanism in a client container it is necessary to

- choose a *callback handler*

  Callback handlers are responsible to get the user identity and to store it.

  The choice of the *callback handler* is done in the `application-xml` file, for example:

```
<application-client>
  <callback-handler>org.ow2.jonas.security.auth.callback.LoginCallbackHandler</callback-
handler>
</application-client>
```

  JOnAS provides several *callback handlers*[8]*:*

  - `LoginCallbackHandler` : it is a text based handler that gets the user and password via stdin
  - `DialogCallbackHandler:` handler using a Swing dialog window to query user and password
  - `NoInputCallbackHandler`: is responsible to store a user/password
  - `CertificateCallback`: is responsible to store a certificate

- configure JASS for setting the LoginModules to be used to perform authentication see Section 2.5.4, “JAAS configuration”

In the `$JONAS_ROOT/examples/javaee5-ear`sample directory can be found examples of clients using JAAS authentication as well as one java client without container client that uses also JAAS.

2.5.4. JAAS configuration

The JAAS configuration is made via the *JAAS Login Configuration File*

A login configuration file consists of one or more entries, each specifying which underlying authentication technology should be used for a particular application or applications.

The contents of the JAAS configuration file has the structure below:

```
Application_1 {
  LoginModuleClassA Flag Options;
  LoginModuleClassB Flag Options;
  LoginModuleClassC Flag Options;
};

Application_2  {
  LoginModuleClassB Flag Options;
  LoginModuleClassC Flag Options;
};

Other {
  LoginModuleClassC Flag Options;
  LoginModuleClassA Flag Options;
};
```

There is a flag associated with all the LoginModules to configure their behaviour in case of success or failure:

- **required** - The LoginModule is required to succeed. If it succeeds or fails, authentication still proceeds through the LoginModule list.
- **requisite** - The LoginModule is required to succeed. If it succeeds, authentication continues through the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed through the LoginModule list).
- **sufficient** - The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed through the LoginModule list). If it fails, authentication continues through the LoginModule list.
- **optional** - The LoginModule is not required to succeed. If it succeeds or fails, authentication still proceeds through the LoginModule list.

2.5.4.1. Default JAAS configuration

JOnAS provides in `$JONAS_BASE/conf/jaas.config` a *JAAS Login Configuration File* already configured with some login configuration.

There are two requirements: the entry dedicated to Tomcat must be named **tomcat**, and the entry for Jetty, **jetty.** Note that everything in this file is case-sensitive.

The predifined entries are:

- **tomcat** used for authentication with the web container Tomcat
- **jetty** used for authentication with the web container Jetty
- **jaasclient** may be used for autentication in a fat client

The default configuration for the web container Tomcat is the following:

```
tomcat {
    org.ow2.jonas.security.auth.spi.JResourceLoginModule required
    resourceName="memrlm_1"
    ;
};
```

this indicates that the `JResourceLoginModule` Login Module must be used on the memory realm named `memrlm_1.`

The default configuration for the web container Jetty is the same than the previous:

```
jetty {
    org.ow2.jonas.security.auth.spi.JResourceLoginModule required
    resourceName="memrlm_1"
    ;
};
```

the configuration for the container clients examples :

```
jaasclient {
    // Login Module to use for the example jaasclient.

    org.ow2.jonas.security.auth.spi.JResourceLoginModule required
    resourceName="memrlm_1"

    org.ow2.jonas.security.auth.spi.ClientLoginModule  required
    globalCtx="true"
    ;
};
```

Here two Login Modules are used, one for checking the identity in the memoty realm, the second for propagating a security context with the client request.

To change the location and name of the `jaas.config` file, edit the `$JONAS_BASE/bin/jonas` script and modify the following line:

```
-Djava.security.auth.login.config=$JONAS_BASE/conf/jaas.config
```

2.5.4.2. JOnAS LoginModules

JOnAS provides some predefined LoginModules:

JResourceLoginModule

This is the main LoginModule. It is highly recommended that this one be used in every authentication, as it checks the user authentication information in the specified realm database, LDAP or memory.

This LoginModule delegates the authentication to the server . Here are the possible attributes to set:

attribute name	description
resourceName	name of the realm
serverName	name of JOnAS instance (default value= jonas)
useUpperCaseUsername	if true Convert the username into uppercase for the authentication (default value=false)
certCallback	if true use certificate callback

CRLLoginModule

This LoginModule contains authentication based on certificates. However, when enabled, it will also permit non-certificate based accesses. It verifies that the certificate presented by the user has not been revoked by the Certification Authority that signed it. To use it, the directory in which to store the revocation lists (*CRLs*) files or an LDAP repository must exist.

attribute name	description
CRLsResourceName	specifies how the *CRLs* are stored:Two possible values "Directory" or "LDAP"
CRLsDirectoryName	The directory containing the *CRL* files (the extension for these files must be `.crl`).
address	address of the server that hosts the LDAP repository
port	port used by the LDAP repository; *CRLs* are retrieved from an LDAP directory using the LDAP schema defined in RFC 2587 [http://www.ietf.org/rfc/rfc2587.txt]

SignLoginModule

login module that signs the current Subject ,. Here are the possible attributes to set:

attribute name	description
keystoreFile	Name of the key store
keystorePass	password for the keystore

keyPass	password for the private key
alias	alias

ClientLoginModule — login module used for propagating the Principal and roles to the server, it doesn't make any authentication. This login module must be used when authentication for a client container. Here is the possible attribute to set:

attribute name	description
globalCtx	if true set the security context for all the threads of the client container instead of only on the current thread. Useful for swing client. (default value= false)

2.6. Configuring JDBC Resource Adapters

Connection of an J2EE application to databases is done through JDBC Resource Adapters (JDBC RA).

Such Resource Adapters are deployed via the **resource** service as seen in Section 2.4.16, “resource service configuration”.

For both container-managed or bean-managed persistence, the JDBC Resource Adapter makes use of relational storage systems through the JDBC interface.

JDBC connections are obtained from a JDBC RA.

The JDBC RA implements the J2EE Connector Specification using the DataSource interface as defined in the JDBC [http://java.sun.com/javase/technologies/database/index.jsp] standard extensions.

An JDBC RA is configured to identify a database and a means to access it via a JDBC driver. Multiple JDBC RAs can be deployed either via the `jonas.properties` file or included in the `autoload` directory of the **resource** service.

The following section explains how JDBC RARs can be defined and configured in the JOnAS server.

To support distributed transactions, the JDBC RA requires the use of at least a JDBC2-XA-compliant driver. Such drivers implementing the XADataSource interface are not always available for all relational databases. The JDBC RA provides a generic driver-wrapper that emulates the XADataSource interface on a regular JDBC driver. It is important to note that this driver-wrapper does not ensure a real two-phase commit for distributed database transactions.

2.6.1. Generic JDBC Resource Adapters

The generic JDBC RAs of JOnAS provide implementations of the `java.sql.Driver`, `javax.sql.DataSource`, `javax.sql.ConnectionPoolDataSource`, and `javax.sql.XADataSource` interfaces. They are located in the `$JONAS_ROOT/rars/autoload` directory and thus are deployed automatically.They consist of base (or generic) RAs facilitating the build of the user JDBC RAs.

Depending on the relational database management server and the available interface in the used JDBC-compliant driver, the user JDBC RA is linked (through the RAR link feature) to a generic RA (for example, the Driver's one). In this case, the user RA contains only a jonas-ra.xml file with some specific parameters, such as the connection url, the user/password, or the JDBC-Driver class.

Resource adapter provided with JOnAS	description	jndi name
rars/autoload/ JOnAS_jdbcDS.rar	Generic JDBC RA that implements the `DataSource` interface	JOnASJDBC_DS
rars/autoload/ JOnAS_jdbcDM.rar	Generic JDBC RA that implements the `Driver` interface	JOnASJDBC_DM
rars/autoload/ JOnAS_jdbcCP.rar	Generic JDBC RA that implements the `ConnectionPoolDataSource` interface	JOnASJDBC_CP
rars/autoload/ JOnAS_jdbcXA.rar	Generic resource adapter that implements the `XADataSource` interface	JOnASJDBC_XA

2.6.2. Specific JDBC Resource Adapter

The remainder of this section, which describes how to define and configure JDBC RAs, is specific to JOnAS. However, the way to use these JDBC RAs in the Application Component methods is standard, i.e., via the resource manager connection factory references (refer to the example in the section Writing Database Access Operations [ejb2_programmer_guide.html#ejb2.bmp].

An RAR file must be deployed as explained in Section 2.4.16, “resource service configuration”.

Usually a resource Adapter contains in its `rar` file all the classes needed to access to the external resource. In the case of a specific JDBC RA it contains only a JOnAS specific deployment descriptor `jonas-ra.xml` that tell what sort of generic resource adapter to use and information related to the specific database used. The `jar` file of the actual JDBC driver mut be copied in the right place to be seen by the JOnAS classloader : `$JONAS_BASE/lib/ext.`

Changing the configuration of the RA requires extracting and editing the deployment descriptor and updating the archive file. There are several possible ways to do this:

- With the RAConfig command (refer to the JOnAS Commands Reference Guide [command_guide.html] for a complete description of the command).
- Through the jonasAdmin console (refer to Administration guide for a complete description). In the jonasAdmin's tree, the Resource Adapter Module node (under the deployment node) contains a configure tab that allows editing of both the ra.xml file and the jonas-ra.xml file of the undeployed RA.

2.6.2.1. Defining the JOnAS Connector Deployment Descriptor: jonas-ra.xml

The `jonas-ra.xml` contains JOnAS specific information describing deployment information, logging, pooling, jdbc connections, and RAR config property values:

- *Deployment Tags*:

property name	description	possible values
jndiname	name the RAR will be registered as. This property is required. This value will be used in the resource-ref section of an Java EE composant.	• Anyname (for example jdbc_1)

rarlink	jndiname of a base RAR file. Useful for deploying multiple connection factories without having to deploy the complete RAR file again. When this is used, the only entry in RAR is a `META-INF/jonas-ra.xml`	• JONASJDBC_DM • JONASJDBC_DS • JONASJDBC_CP • JONASJDBC_XA
native-lib	defines the path where native libraries can be found.	• Any string for a path

- *Logging Tags*:

property name	description	possible values
log-enabled	determines if logging should be enabled for the RAR.	• False (default value) • True
log-topic:	defines the log topic that will be used to write log messages for this rar file.	• Any topic name • Default value is org.objectweb.jonas.jca

- *Pooling Tags*

property name	description	possible values
pool-init	Initial size of the managed connection pool	• 0 (default value) • n
pool-min	Minimum size of the managed connection pool.	• 0 (default value) • n
pool-max	Maximum size of the managed connection pool.	• n • -1 = unlimited (default value)
pool-max-age-minutes	Maximum number of minutes to keep the managed connection in the pool.	• 0 = an unlimited amount of time. • n in minutes
pstmt-max	Maximum number of PreparedStatements per managed connection in the pool. Only needed with the JDBC RA of JOnAS or another database vendor's RAR. Value of 0 is unlimited and -1 disables the cache.	• 0 = unlimited • n (default value = 10) • -1 = cache disabled
pool-max-opentime	Identifies the maximum number of minutes that a managed connection can be left busy.	• 0 = an unlimited amount of time (default value). • n in minutes
pool-max-waiters:	identifies the maximum number of waiters for a managed connection. Default value is 0.	• 0 (default value) • n
pool-max-waittime	identifies the maximum number of seconds that a waiter will	• 0 (default value) • n in seconds

	wait for a managed connection. Default value is 0.	
pool-sampling-period:	identifies the number of seconds that will occur between statistics samplings of the pool. Default is 30 seconds.	• n in seconds (default value = 30s)

- *JDBC Connection Tags*:

Note

Only valid for Connection implementation of java.sql.Connection.

property name	description	possible values
jdbc-check-level	Level of checking that will be done for the jdbc connection.	• 0 : no check (default value) • 1: check connection still open • 2 : send the test statement before reusing a connection from the pool • 3: (keep-alive feature) send the test statement on each connection every pool-sampling-period
jdbc-test-statement	Test SQL statement sent on the connection if the jdbc-check-level is greater than 1.	• A SQL statement

- *Config Property Value Tags:*

  Each entry must correspond to the config-property specified in the ra.xml of the RAR file. The default values specified in the ra.xml will be loaded first and any values set in the jonas-ra.xml will override the specified defaults. These tags differs dependiing on the generic JDBC RA used

property name	description	possible values
dsClass	Name of the class implementing `java.sql.Driver`, `javax.sql.DataSource`, `javax.sql.ConnectionPoolDataSource`, or `javax.sql.XADataSource` interfaces in the JDBC driver.	• any classname representing a JDBC driver (example:org.postgresql.Driver)
URL	Database url of the form jdbc:<database_vendor_subprotocol>. This property may be used only for JDBC RA that implements the Driver (JDBC_DM)	• Any url valid for a database provider (example:jdbc:postgresql://localhost:5432/mydb)
user	Database user name	• any name
password:	Database password	• any string
loginTimeout	Maximum time in seconds that the driver will wait while attempting to connect to a database.	• no value = 0 (default value) • n in seconds

isolationLevel	Level of transaction isolation	• none • serializable • read_committed • read_uncommitted • repeatable_read
mapperName	Name of the JORM mapper	The possible values can be found in the List of available mappers in JORM documentation [http://jorm.objectweb.org/doc/mappers.html].
databaseName	Name of the database	• any name
description:	Informal description	• any String
portNumber	Port Number of the database server	• a number
serverName	Name of the database server.	• any name
dbSpecificMethods	allow flexibility to call setter methods on the dsClass as required by the database provider	see below the particular syntax

- dbSpecificMethods a specific property:

  The JOnAS JDBC Resource Adapter is built as a generic connector to any database provider. The limitation of this is that each database provider may have different requirements about the methods needed to configure the `dataSource` class. This dbSpecificMethods property was added to allow flexibility to call setter methods on the dsClass as required by the database provider. The specific information about what additional methods should be used is documented by the database provider. The format of the value specified is:

  [:<del_char>]<method>=<value>::<value_type>:<method>=<value>::<value_type>....with:

:	optional starting value that denotes using the next character as the delimiter instead of the default ':'
<del_char>	delimiter character to use
<method>	method to call followed by an = sign
<value>	the parameter value to pass to the method being called, followed by 2 delimiter characters.If a Properties object is being passed, then the format of this value must be (name=val, name=val, ...);
<value_type>	the parameter type used to construct the reflection call, followed by the delimiter character if additional methods are being called • Boolean or bool • Byte or byte • Character or char • Double or double

	• Float or float • Integer or int • Long or long • Properties or java.util.Properties • Short or short • String

Note

If this JDBC resource is used as a persistence unit, the persistence configuration defined in the `persistence.xml` file must be coherent to this `jonas-ra.xml` description, such as the datasource name and the dialect.

2.6.2.2. Understanding pooling tags:

At JDBC RA deployment time, if **pool-init** is not null **pool-init** JDBC connection are created.

When a user requests a jdbc connection, the JDBC RA first checks to see if a connection is already open for its transaction. If not, it tries to get a free connection from the free list. If there are no more connections available, it creates a new jdbc connection (if **pool-max** is not reached).

If it cannot create new connections, the user must wait (if **pool-max-waiters** is not reached) until a connection is released. After a limited time (**pool-max-waittime**), the `getConnection` returns an exception.

When the user calls `close()` on its connection, it is put back in the free list.

Many statistics are computed (every **pool-sampling-period** seconds) and can be viewed by JonasAdmin. This is useful for tuning these parameters and for seeing the server load at any time

When a connection has been open for a time too long (**pool-max-age**), the pool will try to release it from the freelist. However, the JDBC RA always tries to keep open at least the number of connections specified in **pool-min**.

When the user has forgotten to close a jdbc connection, the system can automatically close it, after **pool-max-opentime** minutes. Note that if the user tries to use this connection later, thinking it is still open, it will return an exception (socket closed).

When a connection is reused from the freelist, it is possible to verify that it is still valid. This is configured in **jdbc-check-level** . For levels >1 it tries a dummy statement on the connection before returning it to the caller. This statement is configured in **jdbc-test-statement**.

Note

this previous description is not only true for JDBC RAs but also for all types of resource adapters, except **jdbc-check-level** and **jdbc-test-statement** which are specifics for JDBC.

2.6.3. Examples of Specific JDBC Resource Adapter

2.6.3.1. Oracle JDBC resource adapter (Driver)

An RAR for Oracle named as `jdbc_1` in JNDI and using the Oracle thin Driver JDBC driver, should be described in a file (called `for example Oracle1_DM.rar`), with the following properties configured in the jonas-ra.xml file:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<jonas-connector 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-connector_4_2.xsd" >
  <jndi-name>jdbc_1</jndi-name>
  <rarlink>JOnASJDBC_DM</rarlink>
  <jonas-config-property>
    <jonas-config-property-name>user</jonas-config-property-name>
    <jonas-config-property-value>scott</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>password</jonas-config-property-name>
    <jonas-config-property-value>tiger</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>loginTimeout</jonas-config-property-name>
    <jonas-config-property-value></jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>URL</jonas-config-property-name>
    <jonas-config-property-value>jdbc:oracle:thin:@malte:1521:ORA1</jonas-config-property-
value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>dsClass</jonas-config-property-name>
    <jonas-config-property-value>oracle.jdbc.driver.OracleDriver</jonas-config-property-
value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>mapperName</jonas-config-property-name>
    <jonas-config-property-value>rdb.oracle</jonas-config-property-value>
  </jonas-config-property>
</jonas-connector>
```

In this example, "malte" is the hostname of the server running the database Oracle, 1521 is the SQL*Net V2 port number on this server, and ORA1 is the ORACLE_SID. This example makes use of the Oracle "Thin" JDBC driver. For an application server running on the same host as the Oracle DBMS, you can use the Oracle OCI JDBC driver.

2.6.3.2. PostgreSQL JDBC resource adapter (Driver)

To create a PostgreSQL RAR configured as jdbc_3 in JNDI, it should be described in a file (called for examplePostgreSQL3_DM.rar), with the following properties configured in the jonas-ra.xml file:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<jonas-connector 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-connector_4_2.xsd" >
  <jndi-name>jdbc_3</jndi-name>
  <rarlink>JOnASJDBC_DM</rarlink>
<jonas-config-property>
    <jonas-config-property-name>user</jonas-config-property-name>
    <jonas-config-property-value>jonas</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>password</jonas-config-property-name>
    <jonas-config-property-value>jonas</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>loginTimeout</jonas-config-property-name>
    <jonas-config-property-value></jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>URL</jonas-config-property-name>
    <jonas-config-property-value>jdbc:postgresql:/malte:5432/db_jonas</jonas-config-
property-value>
```

```
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>dsClass</jonas-config-property-name>
    <jonas-config-property-value>org.postgresql.Driver</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>mapperName</jonas-config-property-name>
    <jonas-config-property-value>rdb.postgres</jonas-config-property-value>
  </jonas-config-property>
</jonas-connector>
```

2.6.3.3. Oracle JDBC resource adapter (XADataSource)

An RAR for Oracle configured as jdbc_4 in JNDI and using the Oracle `XADataSource` interface of the JDBC driver thin in order to use a JDBC2-XA-compliant driver.It may be described in a file (called for example Oracle1_XA.rar), with the following properties configured in the jonas-ra.xml file:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<jonas-connector 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-connector_4_2.xsd" >
  <jndi-name>jdbc_4</jndi-name>
  <rarlink>JOnASJDBC_XA</rarlink>
   <jonas-config-property>
      <jonas-config-property-name>user</jonas-config-property-name>
      <jonas-config-property-value>jonas</jonas-config-property-value>
   </jonas-config-property>
   <jonas-config-property>
      <jonas-config-property-name>password</jonas-config-property-name>
      <jonas-config-property-value>jonas</jonas-config-property-value>
   </jonas-config-property>
   <jonas-config-property>
      <jonas-config-property-name>databaseName</jonas-config-property-name>
      <jonas-config-property-value>dbjonas</jonas-config-property-value>
    </jonas-config-property>
    <jonas-config-property>
      <jonas-config-property-name>portNumber</jonas-config-property-name>
      <jonas-config-property-value>1521</jonas-config-property-value>
    </jonas-config-property>
    <jonas-config-property>
      <jonas-config-property-name>serverName</jonas-config-property-name>
      <jonas-config-property-value>wallis</jonas-config-property-value>
    </jonas-config-property>
    <jonas-config-property>
      <jonas-config-property-name>dbSpecificMethods</jonas-config-property-name>
      <jonas-config-property-value>:#setDriverType=thin##String</jonas-config-property-
value>
    </jonas-config-property>
    <jonas-config-property>
      <jonas-config-property-name>dsClass</jonas-config-property-name>
      <jonas-config-property-value>oracle.jdbc.xa.client.OracleXADataSource</jonas-config-
property-value>
    </jonas-config-property>
</jonas-connector>
```

2.6.4. Tracing SQL Requests through P6Spy

The P6Spy [http://www.p6spy.com/] tool provides an easy way to trace the SQL requests sent to the database.

To enable this tracing feature, perform the following configuration steps:

- Install the p6spy.jar[9] into $JONAS_BASE/lib/ext.
- Update the appropriate RAR file's `jonas-ra.xml` file by setting the dsClass property to `com.p6spy.engine.spy.P6SpyDriver`
- Set the realdriver property in the `spy.properties` file (located in `$JONAS_BASE/conf`) to the jdbc driver of your actual database.

- Verify that logger.org.objectweb.jonas.jdbc.sql.level is set to DEBUG in $JONAS_BASE/conf/trace.properties.

  Example jonas-ra.xml content:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<jonas-connector 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-connector_4_2.xsd" >
  <jndi-name>jdbc_3</jndi-name>
  <rarlink>JOnASJDBC_DM</rarlink>
  <native-lib></native-lib>
  <log-enabled>true</log-enabled>
  <log-topic>org.objectweb.jonas.jdbc.DMPostgres</log-topic>
  <pool-params>
    <pool-init>0</pool-init>
    <pool-min>0</pool-min>
    <pool-max>100</pool-max>
    <pool-max-age>0</pool-max-age>
    <pstmt-max>10</pstmt-max>
  </pool-params>

 <jdbc-conn-params>
    <jdbc-check-level>0</jdbc-check-level>
    <jdbc-test-statement></jdbc-test-statement>
  </jdbc-conn-params>
  <jonas-config-property>
    <jonas-config-property-name>user</jonas-config-property-name>
    <jonas-config-property-value>jonas</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>password</jonas-config-property-name>
    <jonas-config-property-value>jonas</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>loginTimeout</jonas-config-property-name>
    <jonas-config-property-value></jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>URL</jonas-config-property-name>
    <jonas-config-property-value>jdbc:postgresql://your_host:port/your_db</jonas-config-
property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>dsClass</jonas-config-property-name>
    <jonas-config-property-value>com.p6spy.engine.spy.P6SpyDriver</jonas-config-property-
value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>mapperName</jonas-config-property-name>
    <jonas-config-property-value>rdb.postgres</jonas-config-property-value>
  </jonas-config-property>
  <jonas-config-property>
    <jonas-config-property-name>logTopic</jonas-config-property-name>
    <jonas-config-property-value>org.objectweb.jonas.jdbc.DMPostgres</jonas-config-
property-value>
  </jonas-config-property>
</jonas-connector>
```

  In `$JONAS_BASE/conf/spy.properties` file:

```
realdriver=org.postgresql.Driver
```

  In `$JONAS_BASE/conf/trace.properties`:

```
logger.org.objectweb.jonas.jdbc.sql.level  DEBUG
```

2.6.5. Migration from dbm service to the JDBC RA

The migration of a `Database.properties` file to a similar Resource Adapter can be accomplished through the execution of the following RAConfig tool command. Refer to the JOnAS Commands Reference Guide [command_guide.html#commands.raconfig] for a complete description of RAConfig command.

```
RAConfig -dm -p MySQL1 $JONAS_ROOT/rars/autoload/JOnAS_jdbcDM MySQL_dm
```

Generates a `MySQL_dm.rar` file linked to `JOnAS_jdbcDM.rar`, the `jonas-ra.xml` file inserted is created with values coming from the `ra.xml` file of the `JOnAS_jdbcDM.rar` and values from the `MySQL1.properties` file

The `jonas-ra.xml` created by the previous command can be updated further, if desired. Once the additional properties have been configured, update the `MySQL_dm.rar` file using the following command:

```
RAConfig -path .  MySQL_dm.rar 1

RAConfig -u jonas-ra.xml MySQL_dm.rar 2
```

1 Extraction of `jonas-ra.xml` of `MySQL_dm.rar` in the working directory
2 update `MySQL_dm.rar` with `jonas-ra.xml`

2.7. Configuring JMS Resource Adapters

JMS Resource adapters can be deployed, either via the `jonas.properties` file, or via the JonasAdmin tool, or included in the autoload directory of the **resource** service.

JMS connections are obtained from a JMS RA, which is configured to identify and access a JMS server.

The JORAM resource adapter archive (joram_ra_for_jonas-{joram.version}.rar) is provided with the JOnAS distribution. It is located in the `$JONAS_BASE/repositories/maven2-internal/org/objectweb/joram/joram_ra_for_jonas/{joram.version}/joram_ra_for_jonas-{joram.version}.rar` directory. This file has to be changed if a particular configuration is needed for JORAM.

By default, the joram.xml file, a deployment plan related to JORAM, is present in the `$JONAS_BASE/deploy` directory. This deployment plan is used to deploy JORAM. It declares among others the JORAM resource adapter archive to deploy.

2.7.1. JORAM Resource Adapter configuration files

The JORAM RA may be seen as the central authority to go through for connecting and using a JORAM platform. The RA is provided with a default deployment configuration which:

- Starts a collocated JORAM server in non-persistent mode, with id 0 and name s0, on host localhost and using port 16010; for doing so it relies on both an `a3server.xml` file located in the `$JONAS_BASE/conf` directory and the jonas-ra.xml file located within the RA.
- Creates managed JMS ConnectionFactory instances and binds them with the names **CF**, **QCF**, and **TCF**.
- Creates administered objects for this server (JMS destinations and non-managed factories) as described by the `joramAdmin.xml`, located in the `$JONAS_BASE/conf` directory; those objects are bound with the names **sampleQueue**, **sampleTopic**, **JCF**, **JQCF**, and **JTCF.**

The default configuration may, of course, be modified.

The JORAM integration into JOnAS is composed of 3 different parts: server, RA, and administration. Each part contains its own configuration files:

- a3servers.xml is the JORAM platform configuration file, i.e. the server part. The file is located in the `$JONAS_BASE/conf` directory.

- `ra.xml` and `jonas-ra.xml` are the resource adapter configuration files. They are embedded in the resource adapter (META-INF directory).
- `joramAdmin.xml` contains the administration tasks to be performed by the JORAM server such as the JMS objects creation. It is located in the `$JONAS_BASE/conf` directory.

2.7.1.1. JORAM server configuration : a3servers.xml

The a3server.xml (`$JONAS_BASE/conf/a3server.xml`) file describes the JORAM platform, i.e., the network domain, the used transport protocol, and the reachable JORAM servers. It is used by a JORAM server at start time. By default, only one collocated JORAM server is defined (s0) based on the tcp/ip protocol. A distributed configuration example is provided in the how-to document and other examples are available in JORAM's user guide.

```
<config>
  <property name="Transaction" value="fr.dyade.aaa.util.NullTransaction"/> 1
   <server id="0" name="S0" hostname="localhost"> 2
    <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
             args="root root"/>
    <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
             args="16010"/> 3

   </server>
</config>
```

1 This property means that the non persistent mode for JMS is choosen. In order to use persistent mode, the value must be changed to "fr.dyade.aaa.util.NTransaction"
2 Here can be set the server id and the host where the server run
3 args specifies the port number the JORAM server is listening on

The above configuration describes a JORAM platform made up of one unique JORAM server (id 0, name s0), running on localhost, listening on port 16010. Those values are taken into account by the JORAM server when starting. However, **they should match the values set in the deployment descriptor of the RA**, otherwise the adapter either will not connect to the JORAM server, or it will build improper connection factories.

The joram_raconfig command allows to modify these parameters in all the configuration files.

If used in non-collocated mode, joram can be started with the **JmsServer** command which loads the $JONAS_BASE/conf/a3server.xml configuration file.

2.7.1.2. Resource Adapter configuration: ra.xml, jonas-ra.xml

The `ra.xml` file is the standard deployment descriptor for the JORAM adapter and the `jonas-ra.xml` file is the JOnAS-specific deployment descriptor for the JORAM adapter. These files set the central configuration of the adapter, define and set managed connection factories for outbound communication, and define a listener for inbound communication. `jonas-ra.xml` contains specific parameters such as pool parameters or jndi names, but also may redefine the parameters of some `ra.xml` files and override their values. Globally, a good way to proceed is to keep the original `ra.xml` file with the default values and to customize the configuration only in the `jonas-ra.xml` file.

Changing the configuration of the RA requires extracting and editing the deployment descriptor and updating the archive file. There are several possible ways to do this:

- With the RAConfig command to extract jonas-ra.xml, do the following:

```
RAConfig -path . joram_for_jonas_ra.rar
```

  Then, to update the archive, do the following:

```
RAConfig -u jonas-ra.xml joram_for_jonas_ra.rar
```

- Through the jonasAdmin console (refer to Administration guide for a complete description).

  In the jonasAdmin's tree, the Resource Adapter Module node (under the deployment node) contains a configure tab that allows editing of both the ra.xml file and the jonas-ra.xml file of the undeployed RA.

- Through the joram_raconfig utility (refer to joram_raconfig description for a complete description).

  This tool allows easy modification to the network parameters of the JORAM server in all the configuration files.

The following properties are related to the central configuration of the adapter; they are set via some <jonas-config-property> elements:

property name	description	possible values
CollocatedServer	Running mode of the JORAM server to which the adapter gives access.	• True: when deploying, the adapter starts a collocated JORAM server. • False: when deploying, the adapter connects to a remote JORAM server. • Nothing (default True value is then set).
PlatformConfigDir	Directory where the `a3servers.xml` and `joramAdmin.xml` files are located.	• Any String describing an absolute path (ex: /myHome/myJonasRoot/conf). • Empty String, files will be searched in $JONAS_BASE/conf • Nothing (default empty string is then set).
PersistentPlatform	Persistence mode of the collocated JORAM server. - not taken into account if the JORAM server is set as non-collocated. - If true, set the property 'Transaction' to 'fr.dyade.aaa.util.NTransaction' before launching the JORAM server. - If false, set the property 'Transaction' to 'fr.dyade.aaa.util.NullTransaction' before launching the JORAM server. - Warning, if the 'Transaction' property is set in the a3server.xml file, this value is ignored.	• True: starts a persistent JORAM server. • False: starts a non-persistent JORAM server. • Nothing (default False value is then set).
ServerId	Identifier of the JORAM server to start (not taken into account if the JORAM server is set as non-collocated).	• Identifier corresponding to the server to start described in the a3servers.xml file (ex: 1). • Nothing (default 0 value is then set).

ServerName	Logical name of the JORAM server to start.In the collocated case, this parameter specifies the storage path of the persistent mode (absolute or relative path). If the JORAM server is non-collocated, it must be set to the name of the already started JORAM server (this is necessary for management purpose).	• Storage path of the persistent mode for the collocated case (ex: /tmp/s0). • Name of the started server as described in the a3servers.xml in the non collocated case (ex: s1) • Nothing (default s0 name is then set and the current directory is used for storing the persistent data).
AdminFileXML	Name of the file describing the administration tasks to be performed by the JORAM server, i.e., JMS destinations to create, users to create, ... If the file does not exist, or is not found, no administration task is performed.	• Name of the file (ex: myAdminFile.xml). • Nothing (default joramAdmin.xml name is then set).
HostName	Name of the host where the JORAM server runs, used for accessing a remote JORAM server (non-collocated mode), and for building appropriate connection factories.	• Any host name (ex: myHost). • Nothing (default localhost name is then set).
ServerPort	Port the JORAM server is listening on, used for accessing a remote JORAM server (non-collocated mode), and for building appropriate connection factories.	• Any port value (ex: 16030). • Nothing (default 16010 value is then set).
ConnectingTimer	Duration in seconds during which connecting is attempted (connecting might take time if the server is temporarily not reachable)	• 0 : set for connecting only once and aborting if connecting failed (default value) • n : duration in seconds
CnxPendingTimer	Period in milliseconds between two ping requests sent by the client connection to the server;	• 0 means "notimer" (default value) • n: duration in milliseconds
TxPendingTimer	Duration in seconds during which a JMS transacted (non XA) session might be pending; above that duration the session is rolled back and closed.	• 0 value means "no timer". • n: duration in seconds
DeleteDurableSubscription	Indicates the durable Subscriptions must be deleted when the consumer is closed	• True (previous behaviour) • False (default value)

The <jonas-connection-definition> elements wrap properties related to the managed connection factories:

There are three managed connection factories:

- A Queue managed connection factory registered in JNDI with the name **QCF**
- A Topic managed connection factory registered in JNDI with the name **TCF**
- A managed connection factory registered in JNDI with the name **CF**

Here are the properties that can be configured for each managed connection factory:

property name	description	possible values
jndi-name	Name used for binding the constructed connection factory.	Any name (ex: myQueueConnectionFactory). Default values are • **QCF** for the Queue managed connection factory • **TCF** for the Topic managed connection factory • **CF** for the managed connection factory
UserName	Default user name that will be used for opening JMS connections.	• Any name (ex: myName). • Nothing (default *anonymous* name will be set).
Password	Default user password that will be used for opening JMS connections.	• Any name (ex: myPass). • Nothing (default *anonymous* password will be set).
Collocated	Specifies if the connections that will be created from the factory should be TCP or local-optimized connections	• True (for building local-optimized connections). • False (for building TCP connections). • Nothing (default TCP mode will be set).

The <jonas-activationspec> element wraps a property related to inbound messaging:

property name	description	possible values
jndi-name	Binding name of a JORAM object to be used by 2.1 MDBs.	• Any name (by default:joramActivationSpec).

The Pooling Tags are the same than those for other RAs:

property name	description	possible values
pool-init	Initial size of the managed connection pool	• 0 (default value) • n
pool-min	Minimum size of the managed connection pool.	• 0 (default value) • n
pool-max	Maximum size of the managed connection pool.	• n • -1 = unlimited (default value)

pool-max-age-minutes	Maximum number of minutes to keep the managed connection in the pool.	• 0 = an unlimited amount of time. • n in minutes
pstmt-max	Maximum number of PreparedStatements per managed connection in the pool. Only needed with the JDBC RA of JOnAS or another database vendor's RAR. Value of 0 is unlimited and -1 disables the cache.	• 0 = unlimited • n (default value = 10) • -1 = cache disabled
pool-max-opentime	Identifies the maximum number of minutes that a managed connection can be left busy.	• 0 = an unlimited amount of time (default value). • n in minutes
pool-max-waiters:	identifies the maximum number of waiters for a managed connection. Default value is 0.	• 0 (default value) • n
pool-max-waittime	identifies the maximum number of seconds that a waiter will wait for a managed connection. Default value is 0.	• 0 (default value) • n in seconds
pool-sampling-period:	identifies the number of seconds that will occur between statistics samplings of the pool. Default is 30 seconds.	• n in seconds (default value = 30s)

2.7.1.3. JMS Applications Configuration

`joramAdmin.xml` file describes the configuration related to the application. It describes the administration objects in the JORAM server such as the JMS objects, the users, or the non-managed factories. In other words, it defines the JORAM objects to be (optionally) created when deploying the adapter.

In earlier version the `joram-admin.cfg` was used for this same purpose but it is now deprecated.

The default file provided with JOnAS creates a queue bound with the name *sampleQueue*, a topic bound with the name *sampleTopic*, sets the *anonymous* user, and creates and binds non-managed connection factories named *JCF*, *JQCF* and *JTCF*

Note

- All administration tasks are performed by the server connected but may affect remote JORAM servers to which the adapter is connected through the ServerId attribute.
- If a queue, a topic or a user already exists on the JORAM server (for example, because the server is in persistent mode and has re-started after a crash, or because the adapter has been deployed, undeployed and is re-deployed giving access to a remote JORAM server), it will be retrieved instead of being re-created.

The format of this file is XML. Here are some examples:

- simple example:

```
<?xml version="1.0"?>
```

```
      <JoramAdmin>
        <AdminModule>
          <collocatedConnect name="root" password="root"/>
        </AdminModule>
        <ConnectionFactory
className="org.objectweb.joram.client.jms.tcp.TcpConnectionFactory">
          <tcp host="localhost"
                  port="16010"/>
          <jndi name="JCF"/>
        </ConnectionFactory>
        <ConnectionFactory
className="org.objectweb.joram.client.jms.tcp.QueueTcpConnectionFactory">
          <tcp host="localhost"
                  port="16010"/>
          <jndi name="JQCF"/>
        </ConnectionFactory>
        <ConnectionFactory
className="org.objectweb.joram.client.jms.tcp.TopicTcpConnectionFactory">
          <tcp host="localhost"
                  port="16010"/>
          <jndi name="JTCF"/>
        </ConnectionFactory>
        <User name="anonymous"
                 password="anonymous"
                 serverId="0"/>
        <Queue name="sampleQueue">
          <freeReader/>
          <freeWriter/>
          <jndi name="sampleQueue"/>
        </Queue>
        <Topic name="sampleTopic">
          <freeReader/>
          <freeWriter/>
          <jndi name="sampleTopic"/>
        </Topic>
      </JoramAdmin>
```

- For requesting the creation of a new object, simply add the element in the file. For example, to add a queue 'MyQueue', add the following XML element:

```
        <Queue name="myQueue">
          <freeReader/>
          <freeWriter/>
          <jndi name="myQueue"/>
        </Queue>
```

- When the JORAM is not collocated, the AdminModule must be defined as follows:

```
      <AdminModule>
        <connect host="localhost"
                 port="16020"
                 name="root"
                 password="root"/>
      </AdminModule>
```

  The port number must be set with the server port number (defined in the `a3servers.xml` and in the JORAM's RAR configuration `ra.xml` and `jonas-ra.xml` files).

- Possible parameters for a queue definition:

```
       <Queue name=""
                 serverId=""
                 className=""
                 dmq=""
                 nbMaxMsg=""
                 threshold="">
          <property name="" value=""/>
          <property name="" value=""/>
          <reader user=""/>
          <writer user=""/>
          <freeReader/>
          <freeWriter/>
          <jndi name=""/>
          </Queue>
```

- Possible parameters for a topic definition:

```
<Topic name=""
          parent=""
          serverId=""
          className=""
          dmq="">
  <property name="" value=""/>
  <property name="" value=""/>
  <reader user=""/>
  <writer user=""/>
  <freeReader/>
  <freeWriter/>
  <jndi name=""/>
</Topic>
```

- Example of a dead message queue definition:

```
<DMQueue name="DMQ"
           serverId="0">
  <reader user="anonymous"/>
  <writer user="anonymous"/>
  <freeReader/>
  <freeWriter/>
  <jndi name="DMQ"/>
</DMQueue>
```

- Example of a scheduler queue definition:

```
<Destination type="queue"
               serverId="0"
               name="schedulerQueue"
               className="com.scalagent.joram.mom.dest.scheduler.SchedulerQueue">
  <freeReader/>
  <freeWriter/>
  <jndi name="schedulerQueue"/>
</Destination>
```

- Example of a clustered queues destination:

```
  <Cluster>
    <Queue name="queue0"
             serverId="0"
             className="org.objectweb.joram.mom.dest.ClusterQueue">
      <freeReader/>
      <freeWriter/>
      <property name="period" value="10000"/>
      <property name="producThreshold" value="50"/>
      <property name="consumThreshold" value="2"/>
      <property name="autoEvalThreshold" value="false"/>
      <property name="waitAfterClusterReq" value="1000"/>
      <jndi name="queue0"/>
    </Queue>
    <Queue name="queue1"
             serverId="1"
             className="org.objectweb.joram.mom.dest.ClusterQueue">
      <freeReader/>
      <freeWriter/>
      <property name="period" value="10000"/>
      <property name="producThreshold" value="50"/>
      <property name="consumThreshold" value="2"/>
      <property name="autoEvalThreshold" value="false"/>
      <property name="waitAfterClusterReq" value="1000"/>
      <jndi name="queue1"/>
    </Queue>
    <Queue name="queue2"
             serverId="2"
             className="org.objectweb.joram.mom.dest.ClusterQueue">
      <freeReader/>
      <freeWriter/>
      <property name="period" value="10000"/>
      <property name="producThreshold" value="50"/>
      <property name="consumThreshold" value="2"/>
      <property name="autoEvalThreshold" value="false"/>
      <property name="waitAfterClusterReq" value="1000"/>
      <jndi name="queue2"/>
    </Queue>
    <freeReader/>
    <freeWriter/>
```

```
        <reader user="user0"/>
        <writer user="user0"/>
        <reader user="user1"/>
        <writer user="user1"/>
        <reader user="user2"/>
        <writer user="user2"/>
      </Cluster>
```

2.7.1.4. joram_raconfig command

2.7.1.4.1. joram_raconfig

Change the host and port parameters of a given JORAM server in the configuration files.

2.7.1.4.1.1. Options

`joram_raconfig` [-p port] [-h host] [-s serverId]

-p port	Set the listening port of the JORAM server (defaults to 16010).
-h host	Set the IP address of the JORAM server (defaults to localhost).
-s serverId	Set the server id of the JORAM server (defaults to 0).

2.7.1.4.1.2. Description

The joram_raconfig tool aims to facilitate consistent updates (accross multiple files) for the host and port parameters of a given JORAM server ID.

JORAM relies on several configuration files: `a3servers.xml`, `joramAdmin.xml` and `ra.xml`. With joram_raconfig, these configuration files are updated all together and thus the consistency is ensured.

Modified files:

- `$JONAS_BASE/conf/a3servers.xml`
- `$JONAS_BASE/conf/joramAdmin.xml`
- `META-INF/ra.xml` (in the JORAM resource adapter) is updated.

Resource adapters files are looked up in the following places:

- `$JONAS_BASE/repositories/maven2-internal/org/objectweb/joram/joram_ra_for_jonas/{joram.version}/joram_ra_for_jonas-{joram.version}.rar`
- `$JONAS_BASE/deploy/joram_ra_for_jonas.rar`

2.7.1.4.1.3. Example

```
>$ joram_raconfig -h localhost -p 16012 -s 0
Target JORAM Resource Adapter: /home/ ... /joram/joram_ra_for_jonas/5.2.1a/
joram_ra_for_jonas-5.2.1a.rar
```

2.7.2. JORAM's Resource Adapter tuning

2.7.2.1. ManagedConnection Pool

A pool of ManagedConnection is defined for each factory (connection definition) specified in the jonas-ra.xml file. See the pool parameters in the Section 2.7.1.2, “Resource Adapter configuration: ra.xml, jonas-ra.xml” [63].

2.7.2.2. Session/Thread pool in the JORAM RA

The JORAM RA manages a pool of session/thread for each connection and, by default, the maximum number of parallel sessions is set to 10.

When linked with an message-driven bean, this maximum number of entries in the pool corresponds to the maximum number of messages that can be processed in parallel per message-driven bean. A session is released to the pool just after the message processing (onMessage()). When the maximum is reached, the inquiries for a session creation are blocked until a session becomes available in the pool.

The maxNumberOfWorks property can be set in the message-driven bean standard deployment descriptor. For example, the code below can be added to limit the number of parallel sessions to 100 (default value is 10).

```
<activation-config-property>
<activation-config-property-name>maxNumberOfWorks</activation-config-property-name>
<activation-config-property-value>100</activation-config-property-value>
</activation-config-property>
```

As this parameter set the max number of messages that can be treated simultaneously, the max-cache-size must be set accordingly in the specific deployment descriptor.

2.7.3. Undeploying and Redeploying a JORAM Adapter

Undeploying a JORAM adapter either stops the collocated JORAM server or disconnects from a remote JORAM server. It is then possible to deploy the same adapter again. If set for running a collocated server, it will re-start it. If the running mode is persistent, then the server will be retrieved in its pre-undeployment state (with the existing destinations, users, and possibly messages). If set for connecting to a remote server, the adapter will reconnect and access the destinations it previously created.

In the collocated persistent case, if the intent is to start a brand new JORAM server, its persistence directory should be removed. This directory is located in JOnAS' running directory and has the same name as the JORAM server (for example, s0/ for server "s0").

2.8. Configuring JDBC DataSources

This section describes how to configure the Datasources for connecting application to databases when the **dbm** service is used.

2.8.1. Configuring DataSources

For both container-managed or bean-managed persistence, JOnAS makes use of relational storage systems through the JDBC interface. JDBC connections are obtained from an object, the `DataSource`, provided at the application server level. The `DataSource` interface is defined in the JDBC standard extensions.

A `DataSource` object identifies a database and a means to access it via JDBC (a JDBC driver). An application server may request access to several databases and thus provide the corresponding `DataSource` objects that will be registered in JNDI registry.

This section explains how `DataSource` objects can be defined and configured in the JOnAS server.

JOnAS provides a generic driver-wrapper that emulates the `XADataSource` interface on a regular JDBC driver. It is important to note that this driver-wrapper does not ensure a real two-phase commit for distributed database transactions.

Neither the EJB specification nor the Java EE specification describe how to define `DataSource` objects so that they are available to a Java EE platform. Therefore, this document, which describes how to define and configure DataSource objects, is specific to JOnAS. However, the way to use

these `DataSource` objects in the Application Component methods is standard, that is, by using the resource manager connection factory references (refer to the example in the section Writing database access operations [ejb2_programmer_guide.html#ejb2.bmp] of the Developing Entity Bean Guide [ejb2_programmer_guide.html#ejb2.entity]).

A `DataSource` object should be defined in a file called <DataSource name>.properties (for example Oracle1.properties for an Oracle DataSource or Postgres.properties for an PostgreSQL DataSource.These files must be located in `$JONAS_BASE/conf` directory.

In the `jonas.properties` file, to define a DataSource "Oracle1.properties" add the name "Oracle1" to the line onas.service.dbm.datasources, as follows:

```
jonas.service.dbm.datasources Oracle1, Sybase, PostgreSQL
```

The property file defining a `DataSource` may contain two types of information:

- connection properties
- JDBC Connection Pool properties

2.8.1.1. connection properties

property name	Description
datasource.name	JNDI name of the DataSource
datasource.url	The JDBC database URL : jdbc:<database_vendor_subprotocol>:...
datasource.classname	Name of the class implementing the JDBC driver
datasource.username	Database user name
datasource.password	Database user password
datasource.isolationLevel	Database isolation level for transactions. Possible values are: • none, • serializable, • read_committed, • read_uncommitted, • repeatable_read The default depends on the database used.
datasource.mapper	JORM database mapper (for possible values see here) [http://jorm.objectweb.org/doc/mappers.html]

Note

If this datasource is used as a persistence unit, the persistence configuration defined in the `persistence.xml` file must be coherent to those properties, such as the datasource name and the dialect.

2.8.1.2. Connection Pool properties

Each `Datasource` is implemented as a connection manager and manages a pool of JDBC connections.

The pool can be configured via some additional properties described in the following table.

All these settings have default values and are not required. All these attributes can be reconfigured when JOnAS is running, with the console JonasAdmin.

property	Description	Default value
jdbc.connchecklevel	JDBC connection checking level: • 0 : no check • 1: check connection still open • 2: call the test statement before reusing a connection from the pool	1
jdbc.connteststmt	test statement in case jdbc.connchecklevel = 2.	select 1
jdbc.connmaxage	nb of minutes a connection can be kept in the pool. After this time, the connection will be closed, if minconpool limit has not been reached.	1440 mn (= 1 day)
jdbc.maxopentime	Maximum time (in mn) a connection can be left busy. If the caller has not issued a close() during this time, the connection will be closed automatically.	1440 mn (= 1 day)
jdbc.minconpool	Minimum number of connections in the pool. Setting a positive value here ensures that the pool size will not go below this limit during the datasource lifetime.	0
jdbc.maxconpool	Maximum number of connections in the pool. Limiting the max pool size avoids errors from the database.	no limit
jdbc.samplingperiod	Sampling period for JDBC monitoring. nb of seconds between 2 measures.	60 sec
jdbc.maxwaittime	Maximum time (in seconds) to wait for a connection in case of shortage. This is valid only if maxconpool has been set.	10 sec
jdbc.maxwaiters	Maximum of concurrent waiters for a JDBC Connection. This is valid only if maxconpool has been set.	1000
jdbc.pstmtmax	Maximum number of prepared statements cached in a Connection. Setting this to a bigger value (120 for example) will lead to better performance, but will use more memory. The	12

	recommendation is to set this value to the number of different queries that are used the most often. This is to be tuned by administrators.	

When a user requests a jdbc connection, the **dbm** connection manager first checks to see if a connection is already open for its transaction. If not, it tries to get a free connection from the free list. If there are no more connections available, the **dbm** connection manager creates a new jdbc connection (if jdbc.maxconpool is not reached).

If it cannot create new connections, the user must wait (if jdbc.maxwaiters is not reached) until a connection is released. After a limited time (jdbc.maxwaittime), the `getConnection` returns an exception.

When the user calls `close()` on its connection, it is put back in the free list.

Many statistics are computed (every jdbc.samplingperiod seconds) and can be viewed by JonasAdmin. This is useful for tuning these parameters and for seeing the server load at any time.

When a connection has been open for too long a time (jdbc.connmaxage), the pool will try to release it from the freelist. However, the **dbm** connection manager always tries to keep open at least the number of connections specified in jdbc.minconpool.

When the user has forgotten to close a jdbc connection, the system can automatically close it, after jdbc.maxopentime minutes. Note that if the user tries to use this connection later, thinking it is still open, it will return an exception (socket closed).

When a connection is reused from the freelist, it is possible to verify that it is still valid. This is configured in jdbc.conncheclevel. The maximum level is to try a dummy statement on the connection before returning it to the caller. This statement is configured in jdbc.conntesstmt

2.8.1.3. DataSource example:

Here is the template for an Oracle dataSource.properties file that can be found in $JONAS_ROOT/conf:

```
###################### Oracle DataSource configuration example
#


#####
#  DataSource configuration
#
datasource.name  jdbc_1
datasource.url  jdbc:oracle:thin:@<your-hostname>:1521:<your-db>
datasource.classname oracle.jdbc.driver.OracleDriver
datasource.username <your-username>
datasource.password <user-password>
datasource.mapper rdb.oracle


#####
#  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.conncheclevel 0

#  Max age for jdbc connections
#  nb of minutes a connection can be kept in the pool
jdbc.connmaxage 1440

# Maximum time (in mn) a connection can be left busy.
# If the caller has not issued a close() during this time, the connection
```

```
# will be closed automatically.
jdbc.maxopentime 60

#  Test statement
jdbc.connteststmt select * from dual

# JDBC Connection Pool size.
# Limiting the max pool size avoids errors from database.
jdbc.minconpool 10
jdbc.maxconpool 30

# Sampling period for JDBC monitoring :
# nb of seconds between 2 measures.
jdbc.samplingperiod 30

# Maximum time (in seconds) to wait for a connection in case of shortage.
# This may occur only when maxconpool is reached.
jdbc.maxwaittime 5

# Maximum of concurrent waiters for a JDBC Connection
# This may occur only when maxconpool is reached.
jdbc.maxwaiters 100
```

Chapter 3. EasyBeans Server Configuration File

3.1. Introduction

EasyBeans is configured with the help of an easy-to-understand XML configuration file.

The following is an example of an EasyBeans XML configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<easybeans xmlns="http://org.ow2.easybeans.server">

    <!-- No infinite loop (daemon managed by WebContainer): wait="false"
         Enable MBeans: mbeans="true"
         No EasyBeans naming, use WebContainer naming: naming="false"
         Use EasyBeans JACC provider: jacc="true"
         Use EasyBeans file monitoring to detect archives: scanning="true"
         Use EasyBeans JMX Connector: connector="true"
         Enable Deployer and J2EEServer MBeans: deployer="true" & j2eeserver="true"
    -->
    <config
        wait="false"
        mbeans="true"
        naming="false"
        jacc="true"
        scanning="true"
        connector="true"
        deployer="true"
        j2eeserver="true" />

    <!-- Define components that will be started at runtime -->
    <components>
        <!-- RMI/JRMP will be used as protocol layer -->
        <rmi>
           <protocol name="jrmp" port="1099" hostname="localhost" />
        </rmi>

        <!-- Start a transaction service -->
        <tm />

        <!-- Start a JMS provider -->
        <jms port="16030" hostname="localhost" />

        <!-- Creates an embedded HSQLDB database -->
        <hsqldb port="9001" dbName="jdbc_1">
            <user name="easybeans" password="easybeans" />
        </hsqldb>

        <!-- Add mail factories -->
        <mail>
          <!-- Authentication ?
          <auth name="test" password="test" />
          -->
          <session name="javax.mail.Session factory example" jndiName="mailSession_1">
            <!-- Example of properties -->
            <property name="mail.debug" value="false" />
          </session>

          <mimepart name="javax.mail.internet.MimePartDataSource factory example"
 jndiName="mailMimePartDS_1">
            <subject>How are you ?</subject>
            <email type="to">john.doe@example.org</email>
            <email type="cc">jane.doe@example.org</email>
            <!-- Example of properties -->
            <property name="mail.debug" value="false" />
          </mimepart>

        </mail>

        <!-- Creates a JDBC pool with jdbc_1 JNDI name -->
        <jdbcpool jndiName="jdbc_1" username="easybeans"
            password="easybeans" url="jdbc:hsqldb:hsql://localhost:9001/jdbc_1"
            driver="org.hsqldb.jdbcDriver" />
```

```
        <!-- Start smartclient server with a link to the rmi component-->
        <smart-server port="2503" rmi="#rmi" />


        <!-- JNDI Resolver -->
        <jndi-resolver />

        <!-- JMX component -->
        <jmx />

        <!-- Statistic component -->
        <statistic event="#event" jmx="#jmx" />
    </components>
</easybeans>
```

By default, an `easybeans-default.xml` file is used. To change the default configuration, the user must provide a file named `easybeans.xml`, which is located at classloader/CLASSPATH.

Note

The namespace used is http://org.ow2.easybeans.server.

3.2. Configuration

Each element defined inside the `<components>` element is a component.

Note that some elements are required only for the standalone mode. JMS, RMI, HSQL, and JDBC pools are configured through JOnAS server when EasyBeans runs inside JOnAS.

3.2.1. RMI Component

The RMI configuration is done using the `<rmi>` element.

To run EasyBeans with multiple protocols, the `<protocol>` element can be added more than once.

The hostname and port attributes are configurable.

Protocols could be "jrmp, jeremie, iiop, cmi". The default is jrmp.

Note

Some protocols may require libraries that are not packaged by default in EasyBeans.

3.2.2. Transaction Component

The Transaction Component is defined by the `<tm>` element.

A `timeout` attribute, which is the transaction timeout (in seconds), can be defined on this element. The default is 60 seconds.

The implementation provided by the JOTM [http://jotm.objectweb.org] objectweb project is the default implementation.

3.2.3. JMS Component

The JMS component is used for JMS Message Driven Beans. Attributes are the port number and the hostname.

Also, the workmanager settings can be defined: minThreads, maxThreads and threadTimeout. The values are printed at the EasyBeans startup.

The default implementation is the implementation provided by the JORAM [http://joram.objectweb.org] objectweb project.

3.2.4. HSQL Database

EasyBeans can run an embedded database. Available attributes are the port number and the database name. The `<hsqldb>` may be duplicated in order to run several HSQLDB instances.

Users are defined through the `<user>` element.

3.2.5. JDBC Pool

This component allows the JDBC datasource to be bound into JNDI. The jndi name used is provided by the `jndiName` attribute.

Required attributes are `username`, `password`, `url` and `driver`.

Optional attributes are `poolMin`, `poolMax` and `pstmtMax`. This component provides the option to set the minimum size of the pool, the maximum size, and the size of the prepared statement cache.

3.2.6. Mail component

Mails can be sent by using the mail component that provides either Session or MimePartDataSource factories.

3.2.7. SmartServer Component

This component is used by the Smart JNDI factory on the client side. This allows the client to download missing classes. The client can be run without a big jar file that provides all the classes. Classes are loaded on demand.

Note

Refer to the Chapter titled, Smart JNDI Factory, for more information about this feature.

3.3. Advanced Configuration

This configuration file can be extended to create and set properties on other classes.

3.3.1. Mapping File

A mapping file named `easybeans-mapping.xml` provides the information that rmi is the CarolComponent, tm is the JOTM component, and jms is the Joram component. This file is located in the org.objectweb.easybeans.server package.

The following is an extract of the `easybeans-mapping.xml` file.

Note

The mapping file is using a schema available at http://easybeans.ow2.org/xml/ns/xmlconfig/xmlconfig-mapping_10.xsd [http://easybeans.ow2.org/xml/ns/xmlconfig/xmlconfig-mapping_1_0.xsd]

```
<?xml version="1.0" encoding="UTF-8"?>
 <xmlconfig-mapping xmlns="http://easybeans.ow2.org/xml/ns/xmlconfig"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://easybeans.ow2.org/xml/ns/xmlconfig
                                        http://easybeans.ow2.org/xml/ns/xmlconfig/xmlconfig-
mapping_1_0.xsd">

    <class name="org.ow2.easybeans.server.ServerConfig" alias="config">
        <attribute name="shouldWait" alias="wait" />
        <attribute name="useMBeans" alias="mbeans" />
        <attribute name="useNaming" alias="naming" />
        <attribute name="initJACC" alias="jacc" />
```

```
        <attribute name="directoryScanningEnabled" alias="scanning" />
        <attribute name="startJMXConnector" alias="connector" />
        <attribute name="registerDeployerMBean" alias="deployer" />
        <attribute name="registerJ2EEServerMBean" alias="j2eeserver" />
        <attribute name="description" />
    </class>

    <class name="org.ow2.easybeans.component.Components"
        alias="components" />

    <class name="org.ow2.easybeans.component.util.Property"
        alias="property" />

    <package name="org.ow2.easybeans.component.carol">
        <class name="CarolComponent" alias="rmi" />
        <class name="Protocol" alias="protocol">
          <attribute name="portNumber" alias="port" />
        </class>
    </package>

    <class name="org.ow2.easybeans.component.cmi.CmiComponent" alias="cmi">
        <attribute name="serverConfig" alias="config" />
        <attribute name="eventComponent" alias="event" />
    </class>

    <class
        name="org.ow2.easybeans.component.smartclient.server.SmartClientEndPointComponent"
        alias="smart-server">
        <attribute name="portNumber" alias="port" />
        <attribute name="registryComponent" alias="rmi" />
    </class>

    <class name="org.ow2.easybeans.component.jotm.JOTMComponent"
        alias="tm" />

    <class name="org.ow2.easybeans.component.joram.JoramComponent" alias="jms">
       <attribute name="topic" isList="true" getter="getTopics" setter="setTopics"
element="true"/>
    </class>

    <class
        name="org.ow2.easybeans.component.jdbcpool.JDBCPoolComponent"
        alias="jdbcpool" />

    <class
        name="org.ow2.easybeans.component.remotejndiresolver.RemoteJNDIResolverComponent"
        alias="jndi-resolver">
    </class>

    <package name="org.ow2.easybeans.component.hsqldb">
        <class name="HSQLDBComponent" alias="hsqldb">
            <attribute name="databaseName" alias="dbName" />
            <attribute name="portNumber" alias="port" />
        </class>
        <class name="User" alias="user">
            <attribute name="userName" alias="name" />
        </class>
    </package>

    <package name="org.ow2.easybeans.component.quartz">
        <class name="QuartzComponent" alias="timer" />
    </package>

    <package name="org.ow2.easybeans.component.mail">
      <class name="MailComponent" alias="mail" />
      <class name="Session" alias="session">
        <attribute name="JNDIName" alias="jndiName" />
      </class>
      <class name="MimePart" alias="mimepart">
        <attribute name="subject" element="true" />
        <attribute name="JNDIName" alias="jndiName" />
      </class>
      <class name="MailAddress" alias="email" element-attribute="name" />
      <class name="Auth" alias="auth">
        <attribute name="username" alias="name" />
      </class>
    </package>

    <class name="org.ow2.easybeans.component.event.EventComponent" alias="event">
      <attribute name="eventService" alias="event-service" optional="true" />
    </class>

    <class name="org.ow2.easybeans.component.jmx.JmxComponent" alias="jmx">
```

```
      <attribute name="commonsModelerExtService" alias="modeler-service" optional="true" />
    </class>

    <class name="org.ow2.easybeans.component.statistic.StatisticComponent"
 alias="statistic">
        <attribute name="eventComponent" alias="event" />
        <attribute name="jmxComponent" alias="jmx" />
    </class>

    <package name="org.ow2.easybeans.component.depmonitor">
        <class name="DepMonitorComponent" alias="depmonitor">
        </class>
        <class name="ScanningMonitor" alias="scanning">
            <attribute name="waitTime" alias="period" />
        </class>
        <class name="LoadOnStartupMonitor" alias="loadOnStartup">
        </class>
    </package>


</xmlconfig-mapping>
```

Note

This mapping file is referenced by the easybeans configuration file using the XML namespace : xmlns="http://org.ow2.easybeans.server" .

Each element configured within this namespace will use the mapping done in the `org.ow2.easybeans.server` package.

Users can define their own mapping by providing a file in a package. The name of the the file must be `easybeans-mapping.xml or element-mapping.xml`.

Example: For the element `<easybeans xmlns="http://org.ow2.easybeans.server">`, the resource searched in the classloader is `org/ow2/easybeans/server/easybeans-mapping.xml`. And for an element <pool:max>2</pool:max> with xmlns:pool="http://org.ow2.util.pool.impl", the resource searched will be org/ow2/util/pool/impl/easybeans-mapping.xml or org/ow2/util/pool/impl/pool-mapping.xml.

3.3.2. Other Configuration Files

EasyBeans can be configured through other configuration files as it uses a POJO configuration. If done this way, it can be configured using the Spring Framework component or other frameworks/tools.

Chapter 4. Glossary

Glossary

Axis [http://ws.apache.org/axis/]	Java platform for creating and deploying web services applications
CAROL [http://carol.objectweb.org/]	Library allowing the use of different RMI implementations.
CMI	(Clustered Method Invocation) is the JOnAS cluster protocol for high availability, load-balancing and fail-over
EasyBeans [http://www.easybeans.net/xwiki/bin/view/Main/]	An Open source and lightweight EJB3 container that can be embedded in JOnAS and other application servers. It is an OW2 project.
EIS	Enterprise Information Systems
EJB	Enterprise JavaBeans technology is the server-side component architecture for the Java Platform, Enterprise Edition (Java EE). EJB technology enables rapid development of distributed, transactional, secure and portable applications based on Java technology.
Hibernate	A Java-based object-relational mapping and persistence framework.
IIOP	Inter-operable Internet Object Protocol. It is the CORBA RPC standard protocol on TCP/IP.
JAAS	The Java Authentication and Authorization Service is a set of APIs that enable services to authenticate and enforces access controls upon users.
JACC	Java Authorization Contract for Containers
Jakarta Commons Logging [http://jakarta.apache.org/commons/logging/]	Wrapper around a variety of logging API implementations.
Java EE	Java Platform, Enterprise Edition. A standard for developing portable, robust, scalable and secure server-side Java applications.
JAXP	Java API for XML Processing. Provides the validating and parsing capabilities for XML documents.
JAXR	Java API for XML Registries. Defines a standard API for Java platform applications to access and programmatically interact with different kinds of XML-based metadata registries.
JAX-RPC	Java APIs for XML based RPC.
JAX-WS	Java API for XML-based Web Services. A Java programming language API for creating web services.
J2CA	J2EE Connector Architecture is a standard for facilitating the integration of application servers with heterogeneous Enterprise Information Systems (EISs).

J2EE		Java 2 Platform, Enterprise Edition. A standard for developing portable, robust, scalable and secure server-side Java applications up to version 1.5 of the Java Platform.
JDBC		Java Database Connectivity. The JDBC API provides a call-level API for SQL-based database access.
JDK		The Java Development Kit is set of Java tools (compiler, jvm, library ...) for developing Java programs.
JDO		The Java Data Objects API is a standard interface-based Java model abstraction for persistence.
Jetty	[http://www.mortbay.org/]	A pure java open-source, standards-based, web server implementation.
JGroups	[http://www.jgroups.org/javagroupsnew/docs/index.html]	A toolkit for reliable multicast communication.
JMS		Java Message Service is a Java Message Oriented Middleware (MOM) API.
JMX		Java Management Extensions. A Java technology that supplies tools for managing and monitoring applications.
JNDI		Java Naming Directory Interface. A standard API/SPI for the Java EE naming interface.
JORAM	[http://joram.objectweb.org/]	The Java Open Reliable Asynchronous Messaging is an open source implementation of the JMS API built on top of the ScalAgent [http://www.scalagent.com/] distributed agent technology and hosted by OW2.
JORM	[http://jorm.objectweb.org/]	Java Object Repository Mapping is an OW2 project that provides an adaptable persistence service.
JOTM	[http://jotm.objectweb.org/]	Java Open reliable Transaction Manager is an open source implementation of the JTA APIs hosted by OW2.
JPA		Java Persistence API. A Simpler Programming Model for Entity Persistence.
JSF		JavaServer Faces is a technology that simplifies building user interfaces for JavaServer applications.
JSP		JavaServer Pages is a technology that provides a simplified, fast way to create dynamic web content.
JSTL		JavaServer Pages Standard Tag Library. An extension to the JSP specification that adds a tag library of JSP tags for common tasks, such as, XML data processing, conditional execution, loops and internationalization.
JTA		Java Transaction API. Standard Java interfaces between the transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.
JRE		Java Runtime Environment.

JRMP	Java Remote Method Protocol is a Java RMI standard protocol.
JVM	The Java Virtual Machine.
JWSDL	Java APIs for WSDL. Provides a standard set of Java APIs for representing, manipulating, reading and writing WSDL (Web Services Description Language) documents, including an extension mechanism for WSDL extensibility.
Log4j [http://logging.apache.org/log4j/docs/index.html]	A Java-based logging utility from the Apache Software Foundation. It is used primarily as a debugging tool.
Monolog [http://monolog.objectweb.org/index.html]	The OW2 solution for logging.
MX4J [http://mx4j.sourceforge.net/]	An Open Source implementation of the Java Management Extensions (JMX) and of the JMX Remote API (JSR 160) specifications.
P6Spy [http://www.p6spy.com/]	An open source Java tool that intercepts and logs all database statements that use JDBC.
RMI	Remote Method Invocation. This is the java standard specification for RPC technology.
RPC	Remote Procedure Call is a technology that allows a subroutine or procedure to execute in another address space.
SAAJ	SOAP with Attachments API for Java. Provides a standard way to send XML documents over the Internet from the Java platform.
Speedo [http://speedo.objectweb.org/]	An open source implementation of the JDO 1.0.1 specification hosted by OW2.
Struts [http://struts.apache.org/]	Apache Struts is an open-source framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt the model-view-controller architectural pattern.
Tomcat [http://tomcat.apache.org/]	Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages.
Velocity [http://velocity.apache.org/engine/index.html]	The Apache Velocity Engine is a free open-source templating engine.