ObjectWeb
Open Source Middleware

Getting started with JOnAS 4.8 EE

This is a guide for a first time user of JOnAS. More experiment-ed users that want to learn more may take advantage of reading it..

JOnAS Team (Philippe Coq)

- April 2007 -

Table of Contents

Preface

Welcome to the new users of JOnAS 4.8 EE ! This guide is intended to help you.

Chapter1, First contact with JOnAS 4.8 EE wants to show that a downloaded JOnAS 4.8 EE is usable as it.

The environment to set is truly minimal. It's a child's play to run a Java EE application.

This chapter shows not only how to perform some actions but also explains why all is running so easily.

JOnAS 4.8 EE is distributed with a lot of examples ready to use. Users can learn a lot by studying these examples.

Chapter2, Learning JOnAS by examples wants to help the users to find the answers to some common use cases by indicating which JOnAS example illustrates the use case.

Note that this guide want to stay simple and has not the ambition to resolve all the problems that can be encountered in the *real life* that can be really complex.

For more experimented users that need to perform more complex tasks it is recommended to look the JOnAS 4.8 EE Configuration guide [congfiguration_guide.html]

Appendix A summarize the download and installation instructions

Appendix B describe what you get when you have downloaded and installed JOnAS.

Chapter 1. First contact with JOnAS 4.8 EE

It is assumed, in this guide, that the first time user has already downloaded and installed JOnAS 4.8 EE. If its is not the case, please refer to Appendix A, *Download and installation instructions*. As it is specified in this appendix there are three different distributions. In the rest of this document we presume that the first time user has chosen to download a JOnAS 4.8 EE with a servlet container embedded (for example Apache Tomcat (5.5.17) embedded distribution).

In this chapter an unexperienced user will learn how to run an existing Java EE application with JOnAS, he will understand why it is so easy to achieve such result with nearly zero configuration.

1.1. How do I set up JOnAS environment?

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

- Unix platforms

  Open a new terminal and proceed as follows:

  ```
  bash>export JONAS_ROOT=<your_install_dir>
  bash>export PATH=${PATH}:${JONAS_ROOT}/bin/unix
  ```

  or

  ```
  tcsh>setenv JONAS_ROOT <your_install_dir>
  tcsh>setenv PATH ${PATH}:${JONAS_ROOT}/bin/unix
  ```

- Windows platforms

  Open a new DOS window and proceed as follows:

  ```
  C:>set JONAS_ROOT=C:<your_install_dir>
  C:>set PATH=%PATH%;%JONAS_ROOT%\bin\nt
  ```

  To update the path permanently, do the following depending upon your Windows version:

  Windows XP — Go to the Start Menu, then double click on System. In the System Control Panel select the Advanced tab and push the Environment Variables button. Now, you can look for the PATH to edit. Append the value ;C:\jdk1.4.2\bin (assuming that you installed the SDK in the C:\jdk1.4.2 directory). Once you have changed and saved the value, you will be prompted to reboot.

1.2. How can I check everything is correct?

JOnAS provides a command **jonas** that checks if your environment is set correctly.

Name

jonas check — command allowing to check JOnAS is well installed

Synopsis

```
jonas check
```

Description

Check JOnAS environment is correctly set.

If everything is set correctly, this command ends by displaying something that looks like the following output:

```
JONAS_BASE set to /home/coqp/jb

- JONAS_ROOT value:
/home/coqp/JONAS_4_8_5

- JONAS_BASE value:
/home/coqp/JONAS_4_8_5

- JOnAS Services:
registry,jmx,jtm,db,dbm,security,resource,ejb,ws,web,ear

- Contents of 'jonas.properties':
jonas.service.security.realm.jndi.registration = false
jonas.service.db.user1 = jonas:jonas
jonas.service.mail.factories =
jonas.service.jtm.remote = false
jonas.service.jms.mom = org.objectweb.jonas_jms.JmsAdminForJoram
jonas.service.ejb.auto-genic = true
jonas.service.resource.threadwaittimeout = 60
jonas.service.ejb.descriptors =
jonas.service.mail.class = org.objectweb.jonas.mail.MailServiceImpl
jonas.service.resource.resources =
jonas.service.web.descriptors =
jonas.service.ws.factory.class = org.objectweb.jonas.ws.axis.JAxisServiceFactory
jonas.security.manager = true
jonas.service.ws.wsdlhandlers = file1
jonas.service.ha.jgroups.conf = jgroups-ha.xml
jonas.service.registry.mode = collocated
jonas.service.dbm.datasources = HSQL1
jonas.service.resource.class = org.objectweb.jonas.resource.ResourceServiceImpl
jonas.service.security.class = org.objectweb.jonas.security.JonasSecurityServiceImpl
jonas.service.resource.parsingwithvalidation = true
jonas.service.discovery.class = org.objectweb.jonas.discovery.DiscoveryServiceImpl
jonas.service.ear.descriptors =
jonas.service.thread.class = org.objectweb.area.jonas.AreaService
jonas.service.resource.execworktimeout = 0
jonas.service.registry.class = org.objectweb.jonas.registry.RegistryServiceImpl
jonas.service.jms.queues = sampleQueue
jonas.service.discovery.multicast.address = 224.224.224.224
jonas.service.db.port = 9001
jonas.service.ejb.parsingwithvalidation = true
jonas.service.web.parsingwithvalidation = true
jonas.security.context.check.keystoreFile = /tmp/keystore
jonas.service.thread.ejbareaname = EJB
jonas.service.db.dbname = db_jonas
jonas.service.ejb.autoloaddir = autoload
jonas.service.jms.class = org.objectweb.jonas.jms.JmsServiceImpl
jonas.security.propagation = true
jonas.service.discovery.multicast.port = 9080
jonas.security.context.check = false
jonas.service.thread.file = jonas_areas.xml
jonas.service.discovery.ttl = 1
jonas.service.ha.datasource = jdbc_1
jonas.service.security.ws.realm = memrlm_1
jonas.service.ejb.minworkthreads = 3
jonas.service.dbm.class = org.objectweb.jonas.dbm.DataBaseServiceImpl
jonas.service.ws.wsgen.generator.factory = org.objectweb.jonas_ws.wsgen.generator.ews.EWSGeneratorFactory
jonas.service.ws.class = org.objectweb.jonas.ws.axis.AxisWSServiceImpl
jonas.service.resource.autoloaddir = autoload
jonas.service.jmx.class = org.objectweb.jonas.jmx.JmxServiceImpl
jonas.log.configfile = trace
jonas.service.ejb.maxworkthreads = 80
jonas.service.resource.minworkthreads = 5
jonas.service.web.autoloaddir = autoload
jonas.service.ha.timeout = 600
jonas.service.ha.gcl = jgroups
jonas.service.ear.parsingwithvalidation = true
jonas.service.ear.autoloaddir = autoload
jonas.service.resource.maxworkthreads = 80
jonas.service.ws.parsingwithvalidation = true
jonas.security.context.check.keystorePass = keystorepass
jonas.service.db.class = org.objectweb.jonas.db.hsqldb.HsqlDBServiceImpl
jonas.service.ear.class = org.objectweb.jonas.ear.EarServiceImpl
jonas.service.jtm.class = org.objectweb.jonas.jtm.TransactionServiceImpl
jonas.service.ha.class = org.objectweb.jonas.ha.HaServiceImpl
jonas.service.ejb.threadwaittimeout = 60
jonas.service.ha.jgroups.groupname = jonas-rep
jonas.csiv2.propagation = true
jonas.service.ejb.class = org.objectweb.jonas.container.EJBServiceImpl
jonas.service.jms.topics = sampleTopic
jonas.service.jtm.timeout = 60
jonas.service.jms.collocated = true
jonas.service.security.csiv2.realm = memrlm_1
jonas.service.web.class = org.objectweb.jonas.web.wrapper.catalina55.CatalinaJWebContainerServiceWrapper
jonas.security.context.check.alias = FB
jonas.service.discovery.source.port = 9888
jonas.services = registry,jmx,jtm,db,dbm,security,resource,ejb,ws,web,ear
jonas.transaction.propagation = true

- Contents of 'HSQL1.properties':
jdbc.maxconpool = 100
jdbc.samplingperiod = 30
jdbc.connmaxage = 1440
jdbc.maxwaittime = 5
datasource.url = jdbc:hsqldb:hsql://localhost:9001/db_jonas
datasource.mapper = rdb.hsql
jdbc.minconpool = 10
jdbc.connteststmt = select 1
datasource.username = jonas
datasource.name = jdbc_1
datasource.classname = org.hsqldb.jdbcDriver
jdbc.maxwaiters = 100
datasource.password = jonas
jdbc.connchecklevel = 0
jdbc.maxopentime = 60

- Contents of 'trace.properties':
(file:/home/coqp/JONAS_4_8_5/conf/trace.properties)
handler.logtest.pattern = %d : %l : %h : %O{1}.%M :    %m%n
logger.org.objectweb.jorm.level = WARN
logger.org.objectweb.speedo.level = WARN
logger.root.level = INFO
handler.mesonly.type = Console
logger.org.objectweb.jonas_lib.genbase.level = INFO
logger.org.objectweb.speedo.generation.SpeedoCompiler.level = INFO
logger.org.objectweb.jorm.generator.velocity.level = ERROR
logger.org.objectweb.jotm.level = INFO
logger.org.objectweb.medor.level = WARN
handler.wsdl.pattern = %d : %l : %h : %O{1}.%M :    %m%n
handler.wsdl.output = jonas_wsdls.log
logger.org.objectweb.jonas_ws.wsgen.handler.0 = mesonly
logger.org.objectweb.jonas.genic.handler.0 = mesonly
handler.logtest.type = File
log.config.classname = org.objectweb.util.monolog.wrapper.javaLog.LoggerFactory
logger.org.objectweb.level = INFO
handler.tty.type = Console
handler.logf.output = automatic
logger.org.objectweb.jonas.genic.velocity.level = ERROR
logger.org.objectweb.jonas_tests.history.handler.0 = logtest
logger.org.objectweb.jonas.publication.additivity = false
logger.org.objectweb.jonas.publication.handler.0 = wsdl
handler.tty.output = Switch
handler.logtest.output = jonas_tests.log
handler.logf.pattern = %d : %l : %h : %O{1}.%M :    %m%n
```

```
logger.org.objectweb.jonas_lib.genclientstub.level = INFO
handler.mesonly.pattern = %m%n
logger.org.objectweb.speedo.init.level = INFO
logger.fr.dyade.aaa.level = ERROR
logger.org.objectweb.jonas.genic.level = INFO
logger.org.objectweb.jonas_tests.history.level = INFO
logger.org.objectweb.jonas_ws.wsgen.additivity = false
handler.tty.pattern = %d : %O{1}.%M : %m%n
logger.org.mortbay.util.jmx.ModelMBeanImpl.level = ERROR
logger.org.objectweb.jonas_ws.wsgen.level = INFO
logger.org.objectweb.jonas.jdbc.sql.level = DEBUG
logger.org.objectweb.speedo.mapper.jorm-factory.class-properties.level = INFO
logger.org.objectweb.jonas_tests.history.additivity = false
handler.mesonly.output = Switch
logger.root.handler.1 = logf
logger.root.handler.0 = tty
handler.logf.type = File
logger.org.apache.struts.util.level = WARN
logger.org.objectweb.jonas.genic.additivity = false
handler.wsdl.type = File
logger.org.objectweb.carol.level = INFO
logger.org.objectweb.jonas_ejb.deployment.digester.level = FATAL
logger.org.jgroups.level = FATAL

- Contents of 'carol.properties':
(file:/home/coqp/JONAS_4_8_5/conf/carol.properties)
carol.jeremie.url = jrmi://localhost:2000
carol.irmi.interfaces.bind.single = false
carol.cmi.rr.factor = 100
carol.jrmp.url = rmi://localhost:1099
carol.iiop.server.sslport = 2003
carol.jndi.java.naming.factory.url.pkgs = org.objectweb.jonas.naming
carol.jvm.rmi.local.call = false
carol.irmi.server.port = 0
carol.jrmp.server.port = 0
carol.irmi.url = rmi://localhost:1098
carol.iiop.PortableRemoteObjectClass = org.objectweb.jonas_lib.naming.JacORBPRODelegate
carol.iiop.url = iiop://localhost:2001
carol.iiop.server.port = 0
carol.jrmp.interfaces.bind.single = false
carol.protocols = jrmp
carol.cmi.multicast.groupname = G1
carol.cmi.jgroups.conf = jgroups-cmi.xml
carol.cmi.stub.debug = false
carol.cmi.url = cmi://localhost:2002
carol.jvm.rmi.local.registry = false
carol.jeremie.server.port = 0

- Check 'jonas-realm.xml':
File is present.

- Check 'JORAM configuration':
Ok

The JOnAS environment seems correct.
```

1.3. How can I run a JOnAS server?

Now that your environment seems correct it is possible to launch the JOnAS server simply by typing the following command:

```
jonas start
```

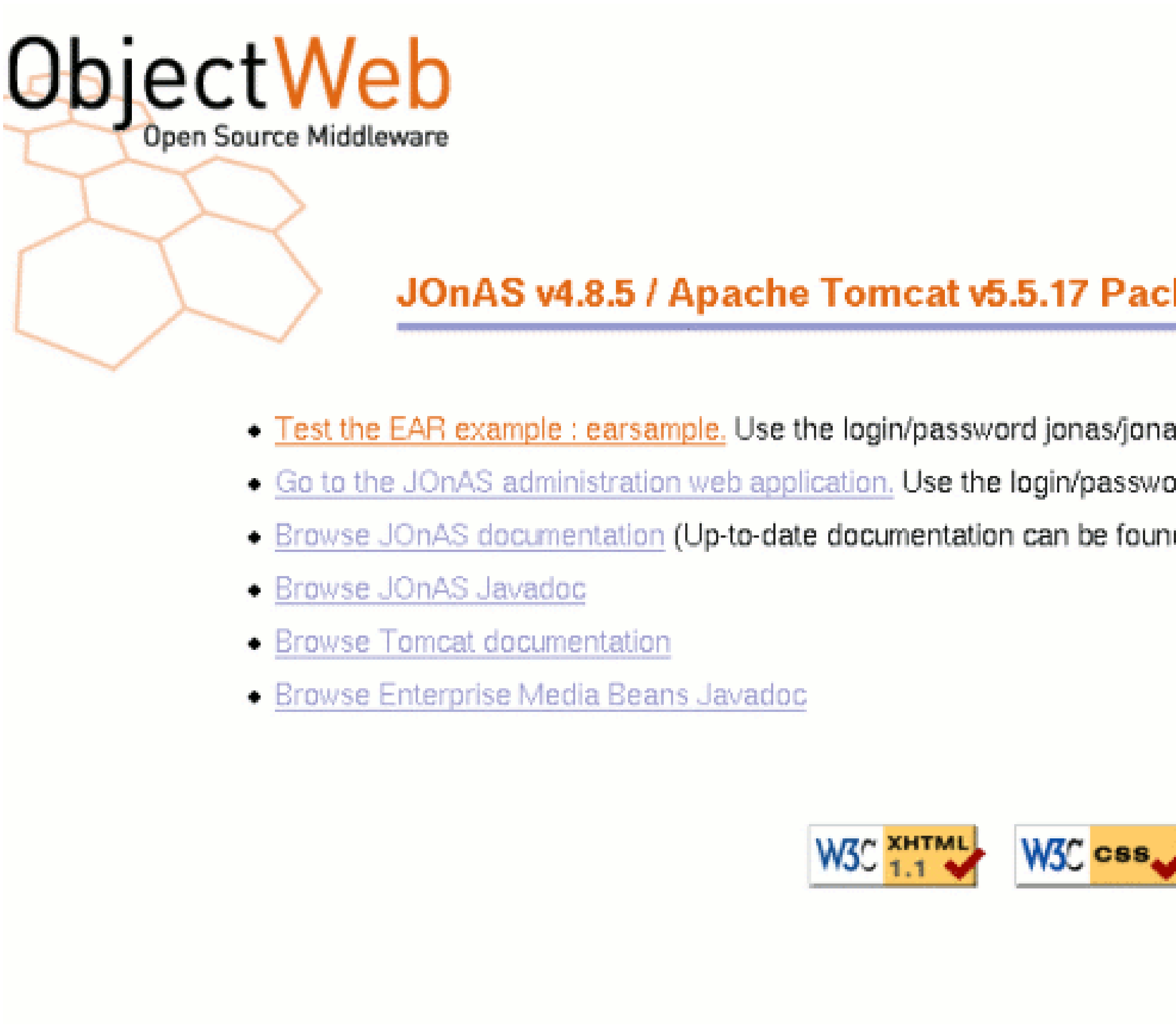
As soon as your server is ready, i.e when you can see on your terminal something that looks like:

```
The JOnAS Server 'jonas' version 4.8.5 is ready
Server$1.run : JOnAS server 'jonas' started on rmi/jrmp
```

You can use your favorite brower and type the following URL:

```
http://localhost:9000/
```

Here is the web page you must get:



JOnAS v4.8.5 / Apache Tomcat v5.5.17 Pac

- Test the EAR example : earsample. Use the login/password jonas/jona
- Go to the JOnAS administration web application. Use the login/passwo
- Browse JOnAS documentation (Up-to-date documentation can be foun
- Browse JOnAS Javadoc
- Browse Tomcat documentation
- Browse Enterprise Media Beans Javadoc

Now, you have just run for the first time JOnAS.

You are now able to do some interesting things like running a sample Java EE application packaged in a `.ear` file (the **earsample** example) or running the web administration tool of JOnAS and some other things.

1.4. How can I run a first Java EE application?

If you have followed the previous steps you are now ready to run your first Java EE application in JOnAS.

Several example programs are included with the JOnAS distribution. They are located in the `$JONAS_ROOT/examples` directory. They are already compiled and ready to use. See in Chapter 2, *Learning JOnAS by examples* for more details.

In a first step we can choose to run the $JONAS_ROOT/examples/earsample example.It is a fairly good examples that shows how to access an EJB deployed in a JOnAS server , from a servlet (thin client).

Note

earsample is in fact a more complex example that can be used to show how the previous EJB can be accessed by a heavy client running in the client container. It can be also used to show how to use security to authentificate the user (in the web container) and how to control user accesses to EJB methods (in the EJB container).

By clicking in the first line of the above web page *Test the EAR example...* you are running a sample Java EE application packaged in a .ear file. It is, in fact, a servlet that:

- get the reference of an stateful session bean
- start some transactions
- performs some works with the stateful session bean
- commit or rollback the started transaction
- display a web page showing all that has been done (this page is ending with **Sample is OK.**)

1.5. Understanding why all this is running

There are several reasons that explain that the previous earsample application is directly runnable on a freshly installed JOnAS:

- Tomcat servlet server is embedded in the distribution
- a web application `ctxroot.war` is pre-installed in `JONAS_ROOT/webapps/autoload` directory. This explains why a web page is displayed when you type : **http://localhost:9000/**
- The **earsample** application is pre-compiled and packaged into a `.ear` file during the building process of the JOnAS distribution
- The `earsample.ear` file is pre-installed in `JONAS_ROOT/apps/autoload` directory . So, this Java EE application will be automaticaly deployed at JOnAS starting time.
- JOnAS is preconfigured: default values are set in configuration files located under `$JONAS_ROOT/conf`.

  These files are accordingly set in order to:

  - force JOnAS to use all the services needed for a correct execution:

    - **registry** service to keep remote references to the session bean home
    - **jtm** service, because the servlet want to start/commit transactions
    - **security** service, because this little application uses EJB security
    - **ejb** service : there is a session bean to be deployed in an ejb container
    - **web** service: there is a servlet to be deployed in a servlet container
    - **ear** service for deploying `earsample.ear` application

    `jonas.properties` is the configuration file

  - set a default port (9000) for the connector HTTP (in server.xml file)

- set a default port (1099) and a default protocol (`jrmp`) to use by registry (in `carol.properties` file)

In Section 1.7, “Running an application with database access” we explain how to run an application that need to access to a database.

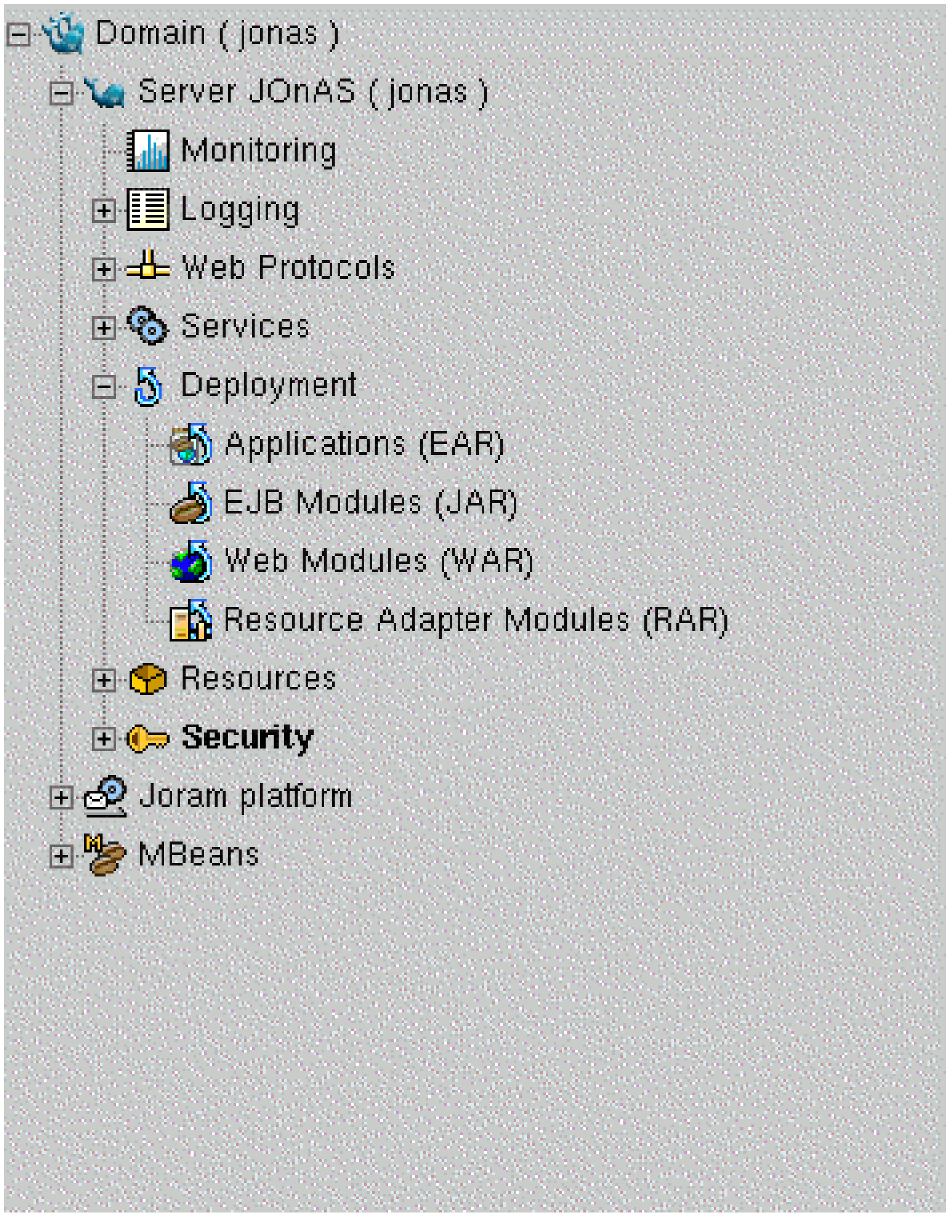
1.6. First step in JOnAS administration

Back to the web page displayed previously, you can notice the second line:

```
Go to the JOnAS administration web application.  Use the login/password jonas/jonas
```

This link allows you to run the JOnAS administration tool **jonasAdmin**.

After the authentication process done (login=jonas,password=jonas) you access to a page in which the left part shows the Management tree:



From this tree it is possible to:

- get information on the management domain
- get information about the JOnAS server (protocol, JMX, registry, servlet server, JVM)
- get monitoring information (threads,memory)
- get or set logging information
- get information on existing web connectors or create new ones
- get information about JOnAS services
- make deployment operations (deploy/undeploy/upload/remove) for `ear`,`war`,`ejbjar`,`rar` files
- get configuration information, statistics, or perform some operations on resources (data sources)
- get configuration information or perform some operations about security
- get configuration information or perform some operations on JORAM JMS provider.
- browse all the deployed MBeans in the server

Here [http://wiki.jonas.objectweb.org/xwiki/bin/download/Main/Demos/clusterManagement.htm]is a demonstration of cluster management features in jonasAdmin console.

1.7. Running an application with database access

Luckily, JOnAS provides a little database, embedded in the server, HSQLDB. This database is not appropriate for real life application but is helpful for running some examples that use entity beans. For using this database the **db** service must be set in the list of services to start, and, it is the case in the JOnAS default configuration.

In the $JONAS_ROOT/examples/src/eb we can find a little example that contains two entity beans that manage `Account` objects. The two beans share the same interface (`Account`); one with bean-managed persistence (BMP, explicit persistence), the other with container-managed persistence (CMP, implicit persistence).

Before running this example, a database table must be created and initialized. We are using Ant (creation and initialization of the database table must be performed after JOnAS is started) :

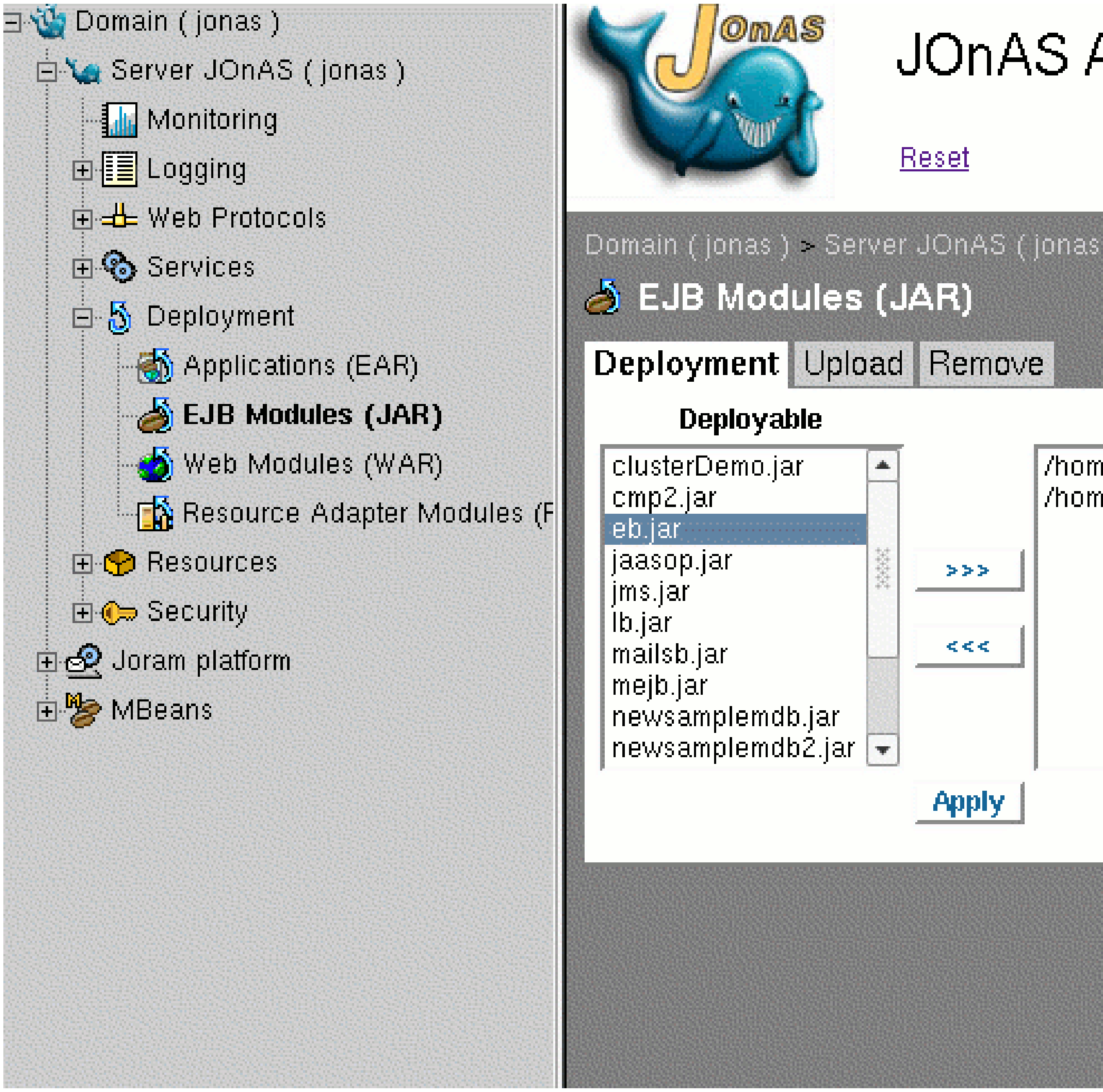
```
bash> cd $JONAS_ROOT/examples/src/eb
bash> ant init_db

init_db:
[java] 1 row updated
[java] 1 row updated
[java] 1 row updated
[java] 1 row updated
[java] 1 row updated
[java] 1 row updated

BUILD SUCCESSFUL
```

As we have yet started JOnAS as indicated in Section 1.3, “How can I run a JOnAS server?” and launched administration tool jonasAdmin as stated in Section 1.6, “First step in JOnAS administration” we are able to deploy the little aplication `eb` which is packaged in an ejbjar file name `eb.jar`.

This deployment task is easily performed via the menu Deployment> EJB Modules (JAR)[1] :



As a result of this operation we can see in the server's terminal:

```
message-header : JContainer.addBean : AccountImpl2 is loaded and using rdb.hsql
message-header : JContainer.addBean : AccountImpl2 available
message-header : JContainer.addBean : AccountImpl available
message-header : JContainer.addBean : AccountExpl available
```

Then, we are able to launch a java client that accesses to the entity beans previously deployed by opening a new terminal and typing :

- Unix platforms

[1]this must be also performed via the **jonas admin -a eb.jar** command

```
bash> jclient -cp $JONAS_BASE/ejbjars/eb.jar:$JONAS_ROOT/examples/classes eb.ClientAccount AccountImpl2Home
```

- Windows platforms

```
C:>jclient -cp %JONAS_BASE%\ejbjars\eb.jar;%JONAS_ROOT%\examples\classes eb.ClientAccount AccountImp2lHome
```

in both cases we must obtain on the terminal something like this:

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

Here again, it is worth to stop a little to detail the *submerged part of Iceberg* in order to understand how this can work:

- in the `$JONAS_ROOT/examples/src/eb` directory are located all the files needed for this application:
  - java files for the EJB Home,EJB Remote and entity bean implementation (compliant to the EJB 2.1 specification) + a java class for a heavy java client that finds all the accounts existing in the database, creates a new one inside a transaction started by the client itself and does some other stuff
  - xml deployment descriptors `eb.xml` and `jonas-eb.xml` respectively the EJB 2.1 standard deployment descriptor and the JOnAS specific one
  - a `build.xml` file for making easy the task of creation of a database table initialized with some rows.
- an ejbjar file (deployment unit) has been build using **ant** and installed under `$JONAS_ROOT/ejbjars` via the target install (look at `$JONAS_ROOT/examples/src/build.xml` file)
- the client class has been compiled via the target compile (look at `$JONAS_ROOT/examples/src/build.xml` file)
- JOnAS is preconfigured: default values are set in configuration files located under `$JONAS_ROOT/conf`.

  These files are accorfingly set in order to:

  - force JOnAS to use all the services needed for a correct execution (`jonas.properties`):
    - **registry** service to keep remote references to the entity bean home object deployed

        - **jtm** service because the heavy client want to start/commit transactions
        - **db** service in order to launch the HSQLDB database server at JOnAS starting time
        - **dbm** service because the client needs to access to a data source (relational database) hosted by HSQLDB
        - **ejb** service because there is an entity bean to deploy in an ejb container
    - set a default port and a default protocol to use by registry in `carol.properties` file (rmi://localhost:1099 protocols=jrmp)
    - make access to the database possible by
        - creating a default database name `db_jonas` via HSQLDB server
        - default configuring an HSQLDB datasource in `HSQL1.properties` with a default connexion url :

          `jdbc:hsqldb:hsql://localhost:9001/db_jonas`
        - providing the corresponding JDBC driver `hsqldb.jar` located in the right classloader $JONAS_ROOT/lib/commons/jonas
- the heavy client has been launched via **jclient**command (which is a wrapper on the **java** command)
    - that set the correct CLASSPATH via a $JONAS_ROOT environment variable correctly set,
    - to which we explicitly give the path of the `location of eb.jar` file and ClientAccount class.

Chapter 2. Learning JOnAS by examples

JOnAS 4.8 EE provides a lot of various examples that use several parts of the Java EE 1.4 specification. It may be interesting for a user that wants to develop a Java EE application and run it, into JOnAS 4.8 EE to look at them to avoid to re-invent the wheel.

2.1. JOnAS examples

We want to describe briefly JOnAS 4.8 EE examples in order to show that they cover a large range of problems more or less complex that may encounter an usual user:

They are located under $JONAS_ROOT/examples and precompiled in the distribution:

earsample	A simple web application packaged as an ear file. A stateful session bean (similar to sb) is called by a servlet. EJB Security is involved. It shows the use of ejb-link, of the local interfaces and illustrates a servlet making use of the `java:comp/` environnement. This example shows also how to add a resource adapter in an ear.Finaly it shows a client running inside the client container and using uniform naming (`java:comp/env`).
alarm	Java EE application using Servlets, JSP session beans, entity beans, message driven beans.It simulates an administration console receiving alarms from some managed devices. The alarms are maintained in database.
cmp2	Shows the new features of container managed persistence in EJB 2.0 like entity relationships and EJB-QL.
src/sb	Example of the use of a stateful session bean (sb) with transactions started on client side. This is the simplest example that is often used to control that JOnAS environment is correct.
src/eb	Example of entity beans (eb) with either container-managed (CMP) or bean-managed persistence (BMP) with transactions started on client side
src/lb	A stateful session bean accessing an entity bean through local interfaces.
src/mailsb	Use of JavaMail API for sending e-mail from a session bean. Two session beans illustrating the use of the two possible mails factories (session and MimePartDataSource).
src/jms	A stateful session bean that sends JMS messages. The java client calls this session bean (for sending a msg on a topic) and creates an eb entity bean (optional) within a tx. A pure JMS client receipts the messages.
src/mdb/samplemdb - src/mdb/newsamplemdb - src/mdb/newsamplemdb2	Message driven bean listening to a JMS topic. The two first differ in the way the JNDI name of the topic is specified : in the stan-

	dard deployment descriptor in the former (EJB2.1 specification) , in the specific jonas deployment descriptor in the later (previous EJB-specification). The third example is equivalent to newsamplemdb but use a JMS queue.
src/mdb/sampleappli	The object of this example is to show how message driven beans may be involved in global transactions as well as entity beans. This application involves an entity bean and two message-driven beans within transactions. It is activated by a simple Java JMS client sending messages to the topic of a message driven bean.
sampleCluster2	Demonstrates the JOnAS's clustering features in 'pedagogic' way. A first level shows the load-balancing at the web level and the second illustrates the load-balancing for the ejb level.This examples is fully described in Clustering demonstration [http://wiki.jonas.objectweb.org/xwiki/bin/view/Main/Demos] on the JOnAS site .
hibernate	Illustrates the use of Hibernate within JOnAS.
jdo	Illustrates the use of JDO (thru speedo) [http://speedo.objectweb.org/] within JOnAS.
j2eemanagement	Sample management application which is using the Management EJB (MEJB) accordingly to the JSR77.
jaasclient	This example shows how to use JAAS login modules for the authentication.There are two two kind of clients: with and without the use of the client container.
olstore	Full e-commerce Java EE application example. There are three versions each one showing the same functionality on a different combination of technologies: • olstore - the original Olstore that uses Struts and EJBs • olstore-spring - Olstore that still uses Struts but communication with the EJBs is simplified using the Spring Framework • olstore-springmvc - in this version the view and controller parts of the application use Spring instead of Struts
petstore1.3	The famous SUN demonstration in its 1.3.2 version.This sample application illustrates design guidelines and architectural recommendations for building distributed applications and represents best practices for many important aspects of Java 2 Platform, Enterprise Edition (Java EE) applications.
webservices/beans/ws	an Application with 1 EJB : a stateless session bean exposed as web service (`ws.ear`). This example also shows how to add security to a Stateless Session bean using **WsGen**. The security parameters are specified in jonas-webservices.xml and the username and password required is jonas:jonas
webservices/beans/wsclient	an Application composed by a session bean acting as web service client accessed through a JSP (`wsclient.ear)`

webservices/webs/webclient	Standalone WebApplication which is client of a WebServices (Google) (`wsaccess.war`)
webservices/webs/webendpoint	Standalone Web Application exposing a JaxRpc Endpoint as WebService and a simple client (`wswarsample.war`)
xdoclet	Example of using xdoclet tags to develop a Java EE application on top of JOnAS

In all this examples can be found a README describing the example contents and how they can be run.

2.2. Examples by topics

Here we try to help user to find which example can illustrate a particular topic. This list is neither exhaustive nor complete.

- **using a java client inside the client container or not**

  see earsample and jaasclient for the use of client container and all examples under src for heavy client without client container

- **clustering**

  see sampleCluster2 and the associated demo [http://wiki.jonas.objectweb.org/xwiki/bin/view/Main/Demos].

- **using EJB 3**

  it is possible to deploy and use EJB3 with JOnAS 4.8 EE + EJB3 EasyBeans [http://www.easybeans.net/xwiki/bin/view/Main/] container. EasyBeans is available for JOnAS 4.x with a resource adapter, you only have to use a JDK 5.0.The guide Getting started with the embedded EasyBeans for JOnAS J2EE server [http://www.easybeans.org/GettingStarted/GettingStarted.html] explains how to do this.

- **transaction management**

  nearly all examples use container managed transactions but some use explicit transaction demarcation in the client (light or heavy) like earsample, src/eb, src/sb

- **servlet and JSP**

  see earsample, alarm, cmp2, olstore, petstore1.3, j2eemanagement, etc...

- **Database connectivity**

  see src/eb, cmp2, alarm, olstore, petstore1.3,hibernate, jdo.

- **JMS and message driven beans**

  src/jms, src/mdb/sampleappli, src/mdb/*, alarm,

- **web services**

  webservices/beans/ws, webservices/beans/wsclient, webservices/webs/webclient, webservices/webs/webendpoint, olstore

- **security**

earsample uses EJB security, jaasclient uses JAAS login module, JACC authentication is used by all servlet examples that use authentication like olstore, petsore1.3,alarm webservices/beans/ws shows how to add security to a Stateless Session bean using WsGen

- **using mail API**

  see src/mailsb.

- **using Java EE management component**

  see j2eemanagement.

- **building and packaging an Java EE application**

  All examples are built and packaged via Ant. Users can be inspired by the build.xml files.

  For example `$JONAS_ROOT/examples/earsample/build.xml` file is a very good example that can be followed. It shows ant targets for:

  - compiling all the java sources
  - building ejbjar file for JOnAS via the standard `ejb` ant target
  - building war file
  - building client jar file
  - building rar file
  - building ear file

Appendix A. Download and installation instructions

A.1. Where do I find JOnAS?

The latest stable binary version can be found on the JOnAS site [http://wiki.jonas.objectweb.org].

The binary versions and sources are available at this site.

JOnAS project is developped under SVN, all information for getting or browsing source code can be found here [http://forge.objectweb.org/plugins/scmsvn/index.php?group_id=5].

A.2. How can I download JOnAS?

The JOnAS download page [http://wiki.jonas.objectweb.org/xwiki/bin/view/Main/Downloads] allows you to choose between :

- JOnAS 4.8 EE with Apache Tomcat (5.5.17) embedded
- JOnAS 4.8 EE with Jetty (5.1.10) embedded
- JOnAS 4.8 EE without web container
- Source package of JOnAS 4.8EE

The three first configurations are available as `.tgz` files or `.exe` auto-installable files for Windows, the fourth only as `.tgz` file.

A.3. How Can I install JOnAS?

The JOnAS distribution can be downloaded as a .tgz or .exe file. It is recommended to download the JOnAS and Tomcat combination or the JOnAS and Jetty combination.

The installation process simply consists of unzipping the downloaded .tgz file or executing the downloaded .exe file.

To install using the .tgz or .exe file select a location for JOnAS installation, for example `your_install_dir`, and point to it.

.

- Unix platforms

```
bash> mkdir -p <your_install_dir>
bash> cd <your_install_dir>
bash> cp <directory-where-jonas-was-downloaded>/<jonas-file-name>.tgz .
bash> tar -zxvf <jonas-file-name>.tgz
```

Caution

Be aware that if a previous version of JOnAS is already installed in this location, the new installation will overwrite previous files and configuration files that have been customized may be lost. In this case, it is recommended that these files be saved before re-starting the installation process.

- Windows platforms

  Open a new DOS window and proceed as follows:

  If the `.tgz` file format was downloaded, you must use a utility such as WinZip or IZArc [http://www.izarc.org/] to extract the files from the archive.

  If the `.exe` file was downloaded, execute the `.exe` file.

A.4. Prerequisites

To be sure JOnAS can be used the following products must be installed:

- a J2SE SDK 1.4 Java virtual machine

  Any J2SE certified java platform may be used to run JOnAS.

  The most commonly used is the SUN's one (Java 2 Platform, Standard Edition [http://java.sun.com/j2se/1.4.2/download.html]), but there is BEA JRockit [http://dev2dev.bea.com/jrockit/], IBM developper kits [http://www-128.ibm.com/developerworks/java/jdk/] or other free/open source certified implementation.

- Ant 1.6 and BCEL

  The binary version of Ant 1.6 must be downloaded from the Ant Apache web site [http://ant.apache.org/bindownload.cgi] and installed

  ```
  bash> tar -jxvf apache-ant-1.6.1-bin.tar.bz2
  or
  bash> unzip apache-ant-1.6.1-bin.zip
  ```

  Set the ANT_HOME environment variable and update the path:

  ```
  bash> export ANT_HOME=<Ant Installation Directory>
  bash> PATH=$PATH;$ANT_HOME/bin
  on Windows :
  C:>set PATH=%ANT_HOME%/bin;%PATH%
  ```

  `bcel-5.1.tar.gz` must be dowloaded from the Jakarta web site [http://www.apache.org/dist/jakarta/bcel/binaries/] then the bcel-5.1.jar must be installed in the `$ANT_HOME/lib` directory.

Appendix B. JOnAS 4.8 EE distribution description

Here is described the tree you get under your installation directory (JONAS_ROOT environment).

In $JONAS_ROOT we find directories and plain files:

Directories are:

- apps

  where Java EE application files (*.ear)* may be installed. The subdirectory *autoload* contains applications that may be deployed at starting time.

- bin

  where are the scripts Unix (in `unix` directory) or .bat windows (in `nt` directory).

- clients

  where Java EE Client Applications (*.jar)* files may be installed.

- conf

  contains JOnAS configuration files.

- demoserver

  this sub tree is an example of JONAS_BASE.The structure of JONAS_BASE is describe in the JOnAS 4.8 EE Configuration guide. [configuration_guide.html]

- ejbjars

  where Enterprise Beans packaged into *ejb-jar* files may be installed.The subdirectory *autoload* contains *ejb-jar* files that may be deployed at starting time.

- examples

  this sub tree contains all the JOnAS examples that are described in Section 2.1, "JOnAS examples"

- the `lib` directory [1]

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

directory	description
`apps`	for apps ClassLoader
`commons`	for the commons ClassLoader
`lib`	the same usage than `commons`
`tools`	for the tools ClassLoader

- logs

  where the log files are created at run-time

- rars

  where ResourceAdapters packaged into `.rar` files may be installed.The subdirectory `autoload` contains ResourceAdapters that will be deployed at starting time.

- templates

  this sub tree contains the following subdirectories used by JOnAS during generation process (it may be code generation, deployment descriptors generation, JONAS_BASE generation.

  - conf is an empty template of JONAS_BASE structure used by tools able to create a JONAS_BASE environment.
  - genic : contains Velocity [http://velocity.apache.org/engine/index.html] templates used by GenIC tool for generating interposition classes
  - newbean: contains Velocity [http://velocity.apache.org/engine/index.html] templates used by the newbean tool
  - resourceadapter: contains Velocity [http://velocity.apache.org/engine/index.html] templates for generating `ra.xml` or `jonas-ra.xml`.
  - wsgen: contains xml descriptors template used by WSGen.

- webapps

  where Web components packaged into `.war` files may be installed.The subdirectory `autoload` contains Web applications that will be deployed at starting time.

- xml

  contains all the DTDs or XML schemas that describe the structure of all the deployment descriptors (standard or specific) for all the JOnAS versions.

  Until EJB2.0 specification (and JOnAS 3.3) DTDs were used, since EJB2.1 specifications and JOnAS 4.0 XML schemas are provided and used to check the validity of the users descriptors.

Appendix C. Glossary

Glossary

Axis [http://ws.apache.org/axis/]	Java platform for creating and deploying web services applications
CAROL [http://carol.objectweb.org/]	Library allowing to use different RMI implementations.
CMI	(Clustered Method Invocation) is the JOnAS protocol cluster 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 ObjectWeb project.
EIS	Enterprise Information Systems
EJB	(Enterprise JavaBeans) technology is the server-side component architecture for 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/persistence framework.
IIOP	(Inter-operable Internet Object Protocol) CORBA RPC standard protocol on TCP/IP.
JAAS	(Java Authentication and Authorization Service) is a set of APIs that enable services to authenticate and enforce access controls upon users.
jakarta commons login [http://jakarta.apache.org/commons/logging/]	Wrapper around a variety of logging API implementations.
Java EE	(Java Platform, Enterprise Edition) standard for developing portable, robust, scalable and secure server-side Java applications.
J2CA	(J2EE Connector Architecture) standard for facilitating the integration of application servers with heterogeneous Enterprise Information Systems (EISs).
J2EE	(Java 2 Platform, Enterprise Edition) standard for developing portable, robust, scalable and secure server-side Java applications.up to version 1.5.
JDBC	(Java Database Connectivity) JDBC API provides a call-level API for SQL-based database access.
JDK	(Java Development Kit) A set a Java tools (compiler, jvm, library ...) for Java programs development.
JDO	(Java Data Objects) API is a standard interface-based Java model abstraction of persistence.

Jetty [http://www.mortbay.org/]	is a pure java open-source, standards-based, web server implemented.
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) is a Java technology that supplies tools for managing and monitoring applications.
JNDI	(Java Naming Directory Interface) Standard API/SPI for Java EE naming interface.
JORAM [http://joram.objectweb.org/]	(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 ObjectWeb
JORM [http://jorm.objectweb.org/]	(Java Object Repository Mapping) is an ObjectWeb project that provide 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 ObjectWeb.
JSP	(JavaServer Pages) is a technology that provides a simplified, fast way to create dynamic web content.
JTA	(Java Transaction API) standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system : the resource manager, the application server, and the transactional applications.
JRE	(Java Runtime Environment).
JRMP	(Java Remote Method Protocol) Java RMI standard protocol.
JVM	(Java Virtual Machine) The Java virtual machine.
Log4j [http://logging.apache.org/log4j/docs/index.html]	is 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]	is the ObjectWeb solution for logging.
MX4J [http://mx4j.sourceforge.net/]	is 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 standard specifications of the Java RPC.

RPC	(Remote Procedure Call) all remote method call protocol is a RPC.
Speedo [http://speedo.objectweb.org/]	is an open source implementation of the JDO 1.0.1 specification hosted by ObjectWeb.
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 a model-view-controller.
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.