Leading Open Source Middleware

Getting started with JOnAS 5

JOnAS Team (Philippe Coq, Guillaume Sauthier)

- March 2009 -

Table of Contents

Preface v
1. First contact with JOnAS 5 1
    1.1. How do I set up a JOnAS environment? 1
        1.1.1. JONAS_BASE creation 1
    1.2. How can I check if everything is correct? 2
        1.2.1. jonas check 2
    1.3. How can I run a JOnAS server? 2
    1.4. How can I run my first Java EE application? 3
    1.5. Understanding how all of this runs out of the box 4
    1.6. First step in JOnAS administration 5
2. Learning JOnAS by examples 6
    2.1. javaee5-earsample example from a 10.000 foot view 6
        2.1.1. Java EE Technologies in Use 6
        2.1.2. javaee5-earsample EAR Sample Architecture 7
    2.2. javaee5-earsample example detailed 9
        2.2.1. JOnAS Server Configuration 9
        2.2.2. Data Tier: JPA Entity Beans 10
        2.2.3. Business Tier: EJB3 Beans 15
        2.2.4. Web Tier: Servlets 25
        2.2.5. Application Clients 29
        2.2.6. EAR 38
A. Download and installation instructions 39
    A.1. Where can I find JOnAS? 39
    A.2. How can I download JOnAS? 39
    A.3. How can I install JOnAS? 39
    A.4. Pre-requisites 39
B. JOnAS 5 distribution description 41
    B.1. JONAS_ROOT structure 41
C. Glossary 43

List of Figures

2.1. javaee5-earsample Sample Architecture 8
2.2. Author and Book JPA Model 10

List of Examples

2.1. JPA Entity: META-INF/persistence.xml 15
2.2. Stateless Session Bean: Reader Business Interface 16
2.3. Stateless Session Bean: Writer business Interface 18
2.4. Stateless Session Bean: Mailer Business Interface 20
2.5. An example of e-mail 30
2.6. META-INF/application-client.xml 33
2.7. META-INF/jonas-client.xml 33
2.8. jaas.config 33
2.9. META-INF/application-client.xml 36
2.10. META-INF/jonas-client.xml 36

Preface

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

Chapter 1, *First contact with JOnAS 5* shows that a downloaded JOnAS 5 is usable as is.

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

This chapter shows not only how to perform some actions but also explains why everything runs so easily.

JOnAS 5 is distributed with examples ready to use. Users can learn a lot by studying these examples

Chapter 2, *Learning JOnAS by examples*

Theses examples may be used as a tutorial for Java EE 5 technology. They are using the simplified programming model, dependency injection, Java Persistence API, security and other functionalities available in Java EE 5.

Note that this guide wants to be simple and is not intended to resolve all problems that can be encountered in *real life* situtations, which can be very complex.

For more experimented users that need to perform more complex tasks, we recommend taking a look at the JOnAS 5 Configuration guide [configuration_guide.html]

Appendix A, *Download and installation instructions* summarizes downloading and installing JOnAS.

Appendix B, *JOnAS 5 distribution description* describes what you get when you download and install JOnAS.

Chapter 1. First contact with JOnAS 5

It is assumed, in this guide, that the first time user has already downloaded and installed JOnAS 5. If this is not the case, please refer to Appendix A, *Download and installation instructions*.

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 results with nearly zero configuration.

1.1. How do I set up a 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.

- Unix platforms

  Open a new terminal and proceed as follows:

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

  or

- Windows platforms

  Open a new DOS window and proceed as follows:

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

  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:\jonas-5.x\bin` (assuming that you installed JOnAS in the `C:\jonas-5.x` directory).

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 to put the configuration files needed by a specific application running on JOnAS in a separate location. This is done by using an additional environment variable called JONAS_BASE.

1.1.1. 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].

1.2. How can I check if everything is correct?

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

1.2.1. jonas check

Command allowing to check that the JOnAS environment is correctly set .

1.2.1.1. Synopsis

`jonas check` [-help]

1.2.1.2. Description

Check that the JOnAS server is well installed. Below the result of the jonas check command :

```
Checking jonas.properties file...
- jonas.services : registry,jmx,jtm,db,security,resource,ejb2,ejb3,jaxws,web,ear,depmonitor

Checking JORAM configuration...

Checking port availability...

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:

```
bash> jonas start

OW2 JOnAS 5.1.0 [ http://jonas.ow2.org / jonas@ow2.org ]

JONAS_BASE is set to xxxxx

Welcome to OW2 JOnAS (Running on Felix).
-----------------------------------------------
```

```
```

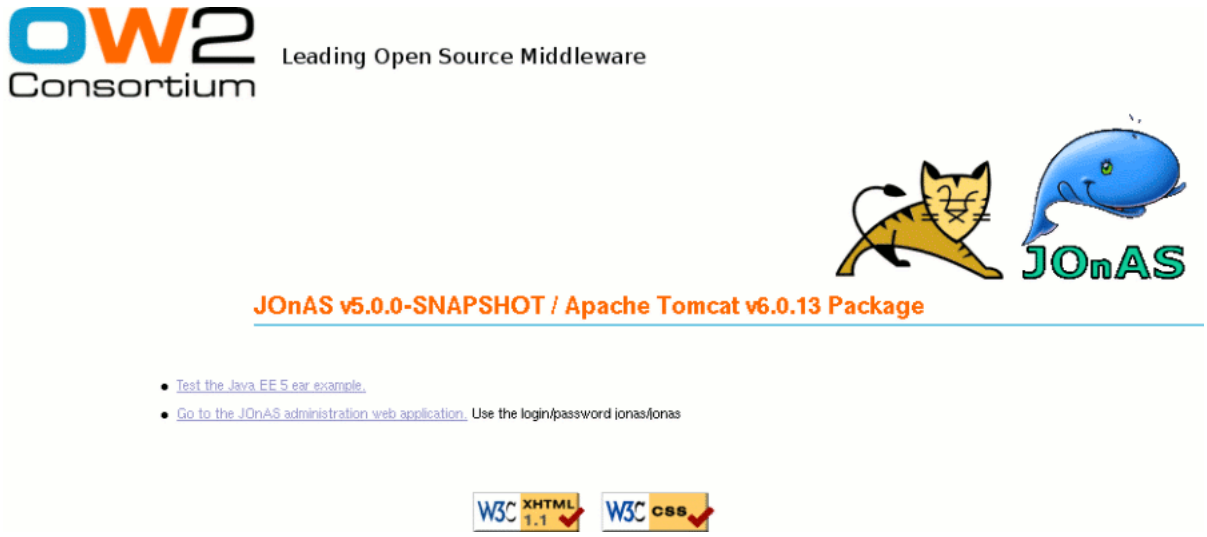
As soon as your server is ready, i.e when you can see on your terminal something that looks like:

```
J2EEServer.__info : JOnAS AS v5.1 named 'jonas' RUNNING
```

you can use your favorite browser and type in the following URL:

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

Here is the web page you should see:



You just have run JOnAS for the first time.

You are now able to do some interesting things like run a sample Java EE application packaged in a `.ear` file (the **earsample** example), run the web administration tool for JOnAS, or you can do some other things that we will explain later on.

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

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

First, we need an application to run. We have chosen the `$JONAS_ROOT/examples/javaee5-earsample` example.It is a fairly good example that shows some interesting functionnalities of the java EE 5 technology. See Chapter 2, *Learning JOnAS by examples* for more details.

You must first compile and generate the Java EE application. This is done by using Ant

```
bash> cd $JONAS_ROOT/examples/javaee5-earsample

bash> ant
```

When you get :

```
Build Successfull
```

You have compiled all the java classes, packaged them in an ear archive and copied it under $JONAS_BASE/deploy.

At JOnAS startup time this Java EE application will be automatically deployed.

Now as soon as your JOnAS server is started you can run the application by typing the following URL in a browser:

```
http://localhost:9000/javaee5-earsample
```

1.5. Understanding how all of this runs out of the box

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

- Tomcat servlet server is embedded in the distribution
- A deployment plan (`ctxroot.xml`) triggering the deployment of the web application `jonas-ctxroot` is pre-installed in `$JONAS_ROOT/deploy` directory. This explains why a web page is displayed when you type: **http://localhost:9000/**
- The `javaee5-earsample.ear` file has been copied in `$JONAS_BASE/deploy` directory. This way, the application has automaticaly been deployed when JOnAS starts.
- By using **newjb** tool JOnAS has been correctly configured and default values has been set in configuration files located under $JONAS_BASE/conf

  These files are accordingly set in order to:

  - force JOnAS to use all the services needed for correct execution:
    - **jtm**: provides the `TransactionManager` used in the back stage (for EJB methods and SQL execution)
    - **db**: provides an HSQL Database for JPA entities
    - **security**: provides security functionalities
    - **resource**: service for using JMS and JDBC resource adaptors
    - **ejb3**: there is a session bean to be deployed in an ejb3 container
    - **web**: there is a servlet to be deployed in a servlet container
    - **ear**: for deploying the `javaee5-earsample.ear` application
    - **jaxws**: for deploying JAX-WS 2.0 web services
    - **depmonitor** that deploys Java EE modules located in `$JONAS_BASE/deploy` directory

    `jonas.properties` is the configuration file
  - set a default port (9000) for the connector HTTP (in `$JONAS_BASE/conf/tomcat6-server.xml` file)
  - set a default port (1099) and a default protocol (`jrmp`) to be used by the registry (in `$JONAS_BASE/conf/carol.properties` file)
  - a JMS resource adaptor correctly configured has been built and installed under $JONAS_BASE/deploy
  - a JDBC resource adaptor for accessing HSQL Database has been built and installed under $JONAS_BASE/deploy

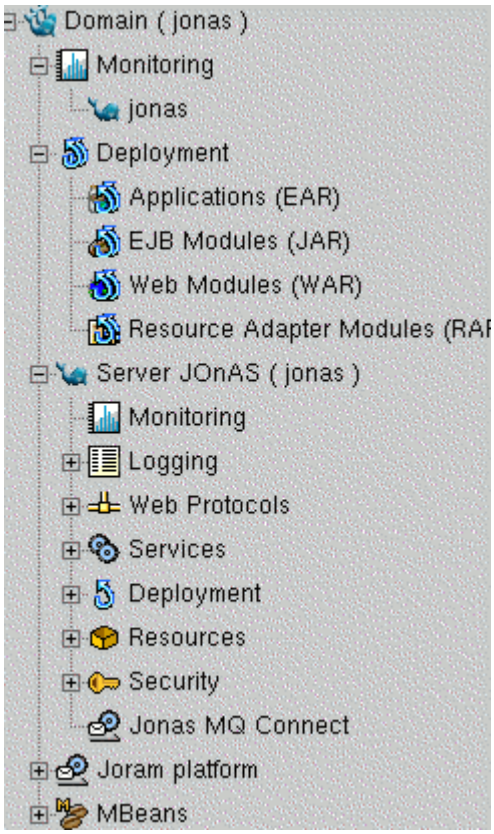
1.6. First step in JOnAS administration

Back on the web page displayed previously, you can see the second line that says:

```
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 is done (login=jonas,password=jonas) you have 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, web container, 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` and `rar` files
- get configuration information, statistics, or perform operations on resources (data sources)
- get securtity configuration information or perform security operations
- get configuration information or perform operations on JORAM, our 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.

Chapter 2. Learning JOnAS by examples

JOnAS 5 provides a complete example **javaee5-earsample** that shows the multitiered application model for enterprise applications that a Java EE platform must support.

In this example you will find all the java EE components possibles:

- Application clients
- Web components (servlets)
- Enterprise JavaBeans™

It may be interesting for a user that wants to start developing a Java EE application and run it in JOnAS 5 to look at the sample in order to see the new Java EE 5 features.

2.1. javaee5-earsample example from a 10.000 foot view

The javaee5-earsample example is a quick-start Java EE application that shows usage of new Java EE 5 features, like JPA, EJB3 (Session and Message Driven Bean), annotations, JAAS security.

JOnAS provides a little database, embedded in the server, namely HSQLDB. This database is helpful for running some examples that use JPA entity beans.

Note

The HSQL database is not appropriate for real life applications.

2.1.1. Java EE Technologies in Use

This sections offers an overview of the different Java EE technologies involved in the javaee5-earsample EAR.

2.1.1.1. JPA 1.0

JPA stands for Java Persistence API, that's a specification extracted from the EJB 3.0 core that addresses persistency problem. Basically, it allows POJOs to be persisted and queried to and from a JDBC database.

Some JPA EntityManager Providers:

- Hibernate [http://www.hibernate.org],
- EclipseLink [http://www.eclipse.org/eclipselink],
- etc ...

2.1.1.2. EJB 3.0

EJB 3.0 is the new Enterprise Java Beans specification [http://java.sun.com/products/ejb/] targeting ease of development (EoD), a feature that was lacking in the previous version (EJB

2.x). This specification makes heavy use of Java 5 annotations (`@Stateless`, `@Stateful`, `@MessageDriven`, ...). One of the main advantages is the introduction of the concept of business method Interceptors.

EJBs are packaged in an EjbJar (`.jar`).

OW2 EasyBeans [http://www.easybeans.net] is the EJB 3.0 Container used in JOnAS 5.

2.1.1.3. JAAS

JAAS is the authentication and authorization mechanism of Java EE that manages Principals and Roles.It is used in this sample to authenticate one of the application's client users.

2.1.1.4. JMS 1.1

JMS is the standard for Messaging system (MOM) in Java. It is used to asynchronously (using JMS Destinations like Queues or Topics) perform calls to the application (usually using MessageDriven Beans).

OW2 JORAM [http://joram.ow2.org] is the JMS provider used in JOnAS 5.

2.1.1.5. Servlet 2.5

Servlets and JSPs compose the web front-end of a classical Java EE application. This is what is accessed when an HTTP URL (http://localhost:9000/javaee5-earsample for example) is called.

Servlets/JSPs are packaged in a web application (`.war`).

2.1.1.6. Application Client

Application Clients are described in the Java EE specification as a way to package code for "heavy-weight" clients (not like web applications, usually known as "light-weight") of the application's business objects (EJBs, JMS Destinations, ...). Application Clients are top level Java EE modules so they benefit from uniform naming (`java:comp/env`).

Application Clients are packaged in `.jar` files.

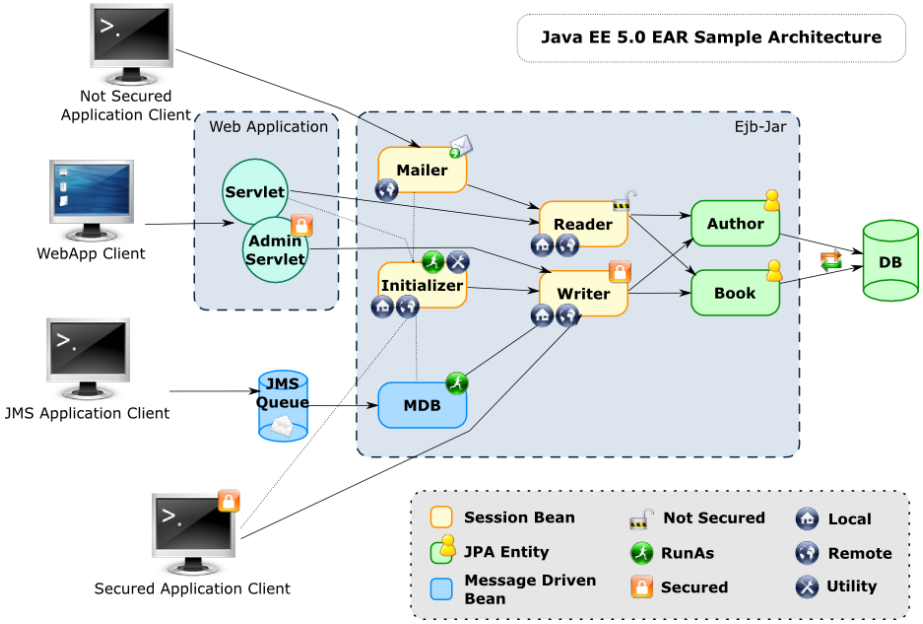
2.1.1.7. Enterprise Application

EARs do not contain any code but are used to package together all the modules composing the application (EjbJar + Web Applications + Resource Adapters + Application Clients).

2.1.2. javaee5-earsample EAR Sample Architecture

In this section, the sample's architecture will be described, execution flow and interactions will be explained.

Figure 2.1. javaee5-earsample Sample Architecture



This application provides the following features:

- 2 separated business interfaces, one allowing write access and the other one read only access.
- It can receive Book creation orders asynchronously from a JMS Queue.
- A Web interface allowing to see the Books and Authors, with a special section authorized to add new Authors.

The database (DB) is provided by a JOnAS service called *db* that launches an Hypersonic SQL server instance. A JDBC Resource Adapter is provided by JOnAS to allow applications to connect to this DB.

Upon the data tier are located the business data objects. They are implemented using the JPA Entity model (with annotations).

The model represented in this application is super easy:

- *Author*: This is the author of a(some) Book(s).
- *Book*: This Entity represents a book written by an Author
- An Author can write many Books
- A Book is written by only 1 Author

Two EJB3 `@Stateless` beans are constructed on top of the business data objects (entities): Reader and Writer.

The *Reader* bean provides read only methods (`findAuthor`, `findBook`, ...) that never change the backed data. On the contrary, the *Writer* bean provides write methods (`addAuthor`, ...) and thus has to be secured.

On top of theses 2 "low level" EJBs, other, more business oriented, EJBs are provided: *Mailer*, *Initializer*, an MDB.

The *Initializer* bean is a "hidden" EJB, whose role is to initialize, if needed, a first set of JPA Entitites (some well known Authors, and some of their Books). This bean is used from almost all the clients to ensure that there is something to show to the user.

The *Mailer* bean can be used by anyone (it's does not use security features) and will only access the *Reader* bean to fill up a mail message that will be sent to a given mail address.

The MDB (MessageDriven Bean) simply creates a new Book for each new *JMS message* received from the JMS Queue.

The following business oriented functions are, in turn, used by the end-user clients:

- Web Application offering a web interface (HTTP/HTML)
- Java EE 5 Application Clients (runs from the command line)

Concerning the web application, it offers 3 pages:

- An index page showing basic instructions on how to use the web application
- An open page (meaning not secured) displaying Books and Authors
- A secured page (needs user authentication) for new Authors addition

There are also multiple standard application clients (AC for short) showing different ways to use the application:

- Not Secured AC: this client uses the Mailer and Reader bean. Using the Reader it displays the Authors and Books, and using the Mailer bean, it sends a mail with an equivalent content.
- JMS AC: this client mainly interacts with a dedicated JMS Queue to sends Book creation orders, then it waits some time and uses the Reader bean to see if the new Books were persisted.
- Secured AC: this clients shows usage of JAAS authentication. It uses the Writer bean, which requires a user with the "earsample" role in order to insert new Authors and Books.

2.2. javaee5-earsample example detailed

This section will focus on multiple aspects of the EAR sample: some basic server configuration (strictly limited to the elements in use for the sample, a more complete description is available in the configuration guide), simple programming guide showing JPA, EJB, JNDI usage...

2.2.1. JOnAS Server Configuration

Here we focus on JOnAS configuration needed by javaee5-earsample examples and obtained after having run the newjb command .

2.2.1.1. Required Services

The javaee5-earsample needs the following JOnAS services to be activated for a flawless execution:

jtm	Provides the transactions support
db	Provides the HSQL Database
mail	Provides the mail sending support
security	Provides the security provider
resource	Provides the connectivity to the database or the JMS provider via resource adapters
ejb3	Provides the EJB 3.0 Container runtime
web	Provides the Web container runtime for web applications (Servlets/JSP)
ear	Provides the Enterprise ARchive (EAR) container

2.2.1.2. DataSource

Access to the dataSource is provided by the resource service .

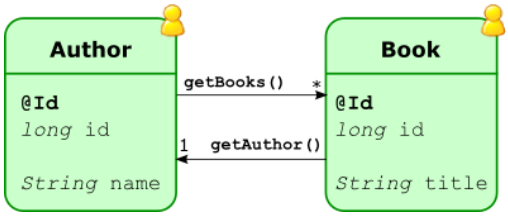
A resource adapter has been created at the JONAS_BASE creation time via newjb tool from the properties found in $JONAS_ROOT/build-jb.properties file:

```
rajdbc.hsql.user=jonas
rajdbc.hsql.password=jonas
rajdbc.hsql.url=jdbc:hsqldb:hsql://localhost:9001/db_jonas  1
rajdbc.hsql.drivername=org.hsqldb.jdbcDriver  2
rajdbc.hsql.jndiname=jdbc_1   3
rajdbc.hsql.jdbcdriverjarfile=none
rajdbc.hsql.mappername=rdb.hsql
```

1. JDBC URL of the connection. This is one is specific to HSQL and corresponds to the HSQL port and DB name provided in the default *db* service configuration.
2. JDBC driver fully qualified class name (including package name)
3. JNDI name of the `DataSource` (`DataSource` will be available under that name in the JNDI registry)

2.2.2. Data Tier: JPA Entity Beans

Figure 2.2. Author and Book JPA Model



This figure represents the model of the application. it is composed of 2 JPA Entities: `Author` and `Book`. Each one has an `id` attribute and are qualified by a `name` and a `title` respectively. Theses entities have a relationship between them: an Author instance knows all the Books he wrote (Author.getBooks()) and a Book knows who wrote it (Book.getAuthor()).

2.2.2.1. Author Bean

A JPA Entity bean is a simple class considered as a POJO (Plain Old Java Object). That means that the class has a public default constructor (no arguments). Moreover, for each attribute/field of the class, a getter and a setter method (`getXYZ()` and `setXYZ(...)`) has to be provided.

A JPA Entity bean will be annotated with the `@Entity` annotation.

The primary key is defined through an `@Id` annotation. In the earsample use case, this key has to be auto-generated. Finally, the `getId()` method should look like this:

```
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    public long getId() {
        return this.id;
    }
```

The Author bean is in relation with the bean Book, because an Author can have many Books (a OneToMany relationship).

The relation is also defined through an annotation:

```
    @OneToMany(mappedBy="author", fetch=FetchType.EAGER, cascade=CascadeType.ALL)
    public Collection<Book> getBooks() {
        return books;
    }
```

This relation is of One-To-Many type. The link will be done with the entity bean Book on the `author` attribute (owner of the relation).

The fetching type is EAGER (it means that when the Author instance is constructed, all the linked books are fetched). Another choice could have been to use LAZY fetching (meaning that the Books will only be fetched when the getBooks() method will be called). In the earsample use case, the entity beans will be accessed from a client in another Java VM (detached mode), so, to reduce latencies caused by network exchanges, it was decided to not use LAZY fetching. Because the Author and Books classes are usable from the remote JVM, they must implements the Serializable interface.

Finally, the cascading mode is set to ALL: when the Author bean is be persisted, all the Books related to the Author bean will be persisted too.

```
package org.ow2.jonas.examples.ear.entity;

import static org.ow2.jonas.examples.ear.entity.Author.QN.ALL_AUTHORS;
import static org.ow2.jonas.examples.ear.entity.Author.QN.FIND_AUTHOR;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collection;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;

/**
 * Author of a book.
 * @author Florent Benoit
 */
@Entity
@NamedQueries({@NamedQuery(name=ALL_AUTHORS, query="select o FROM Author o"),
               @NamedQuery(name=FIND_AUTHOR, query="select o FROM Author o WHERE name
 = :name")})
public class Author implements Serializable {
    /**
     * Store Query names.
     */
    public static interface QN {
        /**
         * Search all authors.
         */
        String ALL_AUTHORS = "Author.allAuthors";

        /**
         * Search a named author.
         */
        String FIND_AUTHOR = "Author.findAuthor";
    }

    /**
     * Serial Version UID.
     */
    private static final long serialVersionUID = 0L;

    /**
     * Primary key (will be auto generated).
     */
    private long id;

    /**
     * Name of the author.
     */
    private String name = null;

    /**
     * List of books written by the author.
     */
    private Collection<Book> books;

    /**
     * Default constructor.
     */
    public Author() {
        books = new ArrayList<Book>();
    }
```

```
    /**
     * Constructor with a given author name.
     * @param name - the name of the author
     */
    public Author(final String name) {
        this();
        setName(name);
    }

    /**
     * Relation ship (do not using lazy mode).
     * @return books written by this author
     */
    @OneToMany(mappedBy="author", fetch=FetchType.EAGER, cascade=CascadeType.ALL)
    public Collection<Book> getBooks() {
        return books;
    }

    /**
     * Add a book with a given title.
     * @param title - the title of the book
     */
    public void addBook(final String title) {
        Book livre = new Book();
        livre.setTitle(title);
        livre.setAuthor(this);
        getBooks().add(livre);
    }

    /**
     * Sets the collection of books written by this author.
     * @param books the list of the books
     */
    public void setBooks(final Collection<Book> books) {
        this.books = books;
    }

    /**
     * @return name of the author
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the name of the author.
     * @param name - the name of this author
     */
    public void setName(final String name) {
        this.name = name;
    }

    /**
     * @return an id for this object (incremented automatically)
     */
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    public long getId() {
        return this.id;
    }

    /**
     * Sets the id of this author object.
     * @param id the given id of this author
     */
    public void setId(final long id) {
        this.id = id;
    }


    /**
     * @return String representation of this entity object.
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder(this.getClass().getName());
        sb.append("[id=");
        sb.append(getId());
        sb.append(", name=");
        sb.append(getName());
        sb.append("]");
        return sb.toString();
```

```
    }
}
```

Note

Two EJB-QL queries have been defined in the bean's class.They will be used to retrieve all the authors and an author with a given name respectively.

2.2.2.2. Book Bean

The Book bean is also a JPA entity bean. There is a relationship with the Author bean, one book being related to only one Author: Many-To-One relationship.

The @JoinColumn annotation defines the column of the primary key to be used for the association between 2 entity beans:

```
    @ManyToOne
    @JoinColumn(name="Author_id")
    public Author getAuthor() {
        return author;
    }
```

Here is the complete Book class:

```
package org.ow2.jonas.examples.ear.entity;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;

/**
 * Define a book.
 * @author Florent Benoit
 */
@Entity
@NamedQueries({@NamedQuery(name=Book.QN.ALL_BOOKS, query="select o FROM Book o"),
               @NamedQuery(name=Book.QN.FIND_BOOK, query="select o FROM Book o WHERE name
 = :name")
})
public class Book implements Serializable {

    /**
     * Defines Query names.
     */
    public static interface QN {
        /**
         * Search all books.
         */
        String ALL_BOOKS = "Book.allBooks";

        /**
         * Search a book.
         */
        String FIND_BOOK = "Book.findBook";
    }

    /**
     * Serial Version UID.
     */
    private static final long serialVersionUID = 0L;

    /**
     * Primary key.
     */
    private long id;

    /**
     * Author's book.
     */
```

```
    private Author author;

    /**
     * title of the book.
     */
    private String title;

    /**
     * Default constructor.
     */
    public Book() {

    }

    /**
     * Constructor. Build a new Book with the given title and written by the
     * given author.
     * @param title the given title
     * @param author the given author.
     */
    public Book(final String title, final Author author) {
        setTitle(title);
        setAuthor(author);
    }

    /**
     * @return the Author of this Book.
     */
    @ManyToOne
    @JoinColumn(name="Author_id")
    public Author getAuthor() {
        return author;
    }

    /**
     * Sets the author of this book.
     * @param author the given author.
     */
    public void setAuthor(final Author author) {
        this.author = author;
    }

    /**
     * @return the title of this book.
     */
    public String getTitle() {
        return title;
    }

    /**
     * Set the title of the book.
     * @param title - the title of the book
     */
    public void setTitle(final String title) {
        this.title = title;
    }

    /**
     * @return an id for this object (incremented automatically)
     */
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    public long getId() {
        return this.id;
    }

    /**
     * Sets the id of this author object.
     * @param id the given id of this author
     */
    public void setId(final long id) {
        this.id = id;
    }

    /**
     * @return String representation of this entity object.
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder(this.getClass().getName());
        sb.append("[id=");
        sb.append(getId());
        sb.append(", title=");
```

```
        sb.append(getTitle());
        sb.append("]");
        return sb.toString();
    }
}
```

2.2.2.3. Persistence File

It is required to have a persistency file `META-INF/persistence.xml` to describe useful information. For example, the JNDI name of the `DataSource` to be used for database persistency.

Example 2.1. JPA Entity: META-INF/persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
<persistence-unit name="entity" transaction-type="JTA">
   <provider></provider>
   <jta-data-source>jdbc_1</jta-data-source>
   <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>

      <property name="eclipselink.ddl-generation" value="drop-and-create-tables"/>
      <property name="eclipselink.ddl-generation.output-mode" value="database"/>
      <property name="eclipselink.target-database" value="HSQL"/>

      <property name="toplink.target-database" value="HSQL"/>
      <property name="toplink.ddl-generation" value="drop-and-create-tables"/>
      <property name="toplink.ddl-generation.output-mode" value="database"/>

      <property name="openjpa.jdbc.DBDictionary" value="hsql"/>
      <property name="openjpa.jdbc.SynchronizeMappings"
 value="buildSchema(ForeignKeys=true)"/>
   </properties>
</persistence-unit>
</persistence>
```

2.2.3. Business Tier: EJB3 Beans

This section will decribe the business parts of the application. Session beans provide synchronous RPC interfaces and the MessageDriven bean provides the asynchronous interface.

2.2.3.1. Session Beans

The Session beans have been separated into two logical parts: Writer and Reader beans provide the basic operations (read/write) on the entities, while Mailer and Initializer are using the previous ones to perform more advanced operations.

2.2.3.1.1. Reader Bean

The Reader Bean is a Stateless Session Bean that provides both a Remote and a Local business interface. It is responsible for all the read-like operations the application can do on the JPA entities. The `listAllXYZ` and `findXYZ` methods return the JPA entity beans.

Example 2.2. Stateless Session Bean: Reader Business Interface

```
package org.ow2.jonas.examples.ear.reader;

import java.util.List;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;

/**
 * The {@link Reader} business interface is an un-restricted
 * read only view of the entities.
 * @author Guillaume Sauthier
 */
public interface Reader {

    /**
     * @return the list of all the persisted {@link Author}s.
     */
    List<Author> listAllAuthors();

    /**
     * @return the list of all the persisted {@link Book}s.
     */
    List<Book> listAllBooks();

    /**
     * Find a given {@link Author} using it's name as a key.
     * @param name {@link Author}'s name.
     * @return the first {@link Author} that matches the given name.
     */
    Author findAuthor(final String name);

    /**
     * Find a given {@link Book} using it's name as a key.
     * @param name {@link Book}'s name.
     * @return the first {@link Book} that matches the given name.
     */
    Book findBook(final String name);
}
```

This bean uses an `EntityManager` (injected through the `@PersistenceContext` annotation).

```
    /**
     * Entity manager used by this bean.
     */
    @PersistenceContext
    private EntityManager entityManager = null;
```

The EntityManager is also used to perform EJB-QL queries (they have been defined with the @NamedQuery annotation on top of the JPA classes).

```
    /**
     * @returns the list of all the persisted {@link Author}s.
     */
    public List<Author> listAllAuthors() {
        return entityManager.createNamedQuery(ALL_AUTHORS).getResultList();
    }
```

Here is the code of the Reader EJB:

```
package org.ow2.jonas.examples.ear.reader;

import static org.ow2.jonas.examples.ear.entity.Author.QN.ALL_AUTHORS;
import static org.ow2.jonas.examples.ear.entity.Author.QN.FIND_AUTHOR;
import static org.ow2.jonas.examples.ear.entity.Book.QN.ALL_BOOKS;
import static org.ow2.jonas.examples.ear.entity.Book.QN.FIND_BOOK;

import java.util.List;

import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
```

```
import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;

/**
 *The {@link ReaderBean} EJB is an unrestricted, read-only, Stateless Bean.
 * @author Guillaume Sauthier
 */
@Stateless
@Local(LocalReader.class)
@Remote(RemoteReader.class)
public class ReaderBean implements LocalReader, RemoteReader {

    /**
     * Entity manager used by this bean.
     */
    @PersistenceContext
    private EntityManager entityManager = null;

    /**
     * Find a given {@link Author} using it's name as a key.
     * @param name {@link Author}'s name.
     * @return the first {@link Author} that matches the given name.
     */
    @SuppressWarnings("unchecked")
    public Author findAuthor(final String name) {
        Query query = entityManager.createNamedQuery(FIND_AUTHOR);
        query.setParameter("name", name);
        List<Author> authors = query.getResultList();
        if (authors != null && authors.size() > 0) {
            return authors.get(0);
        }
        return null;
    }

    /**
     * Find a given {@link Book} using it's name as a key.
     * @param name {@link Book}'s name.
     * @return the first {@link Book} that matches the given name.
     */
    @SuppressWarnings("unchecked")
    public Book findBook(final String name) {
        Query query = entityManager.createNamedQuery(FIND_BOOK);
        query.setParameter("name", name);
        List<Book> books = query.getResultList();
        if (books != null && books.size() > 0) {
            return books.get(0);
        }
        return null;
    }

    /**
     * @return the list of all the persisted {@link Author}s.
     */
    @SuppressWarnings("unchecked")
    public List<Author> listAllAuthors() {
        return entityManager.createNamedQuery(ALL_AUTHORS).getResultList();
    }

    /**
     * @return the list of all the persisted {@link Book}s.
     */
    @SuppressWarnings("unchecked")
    public List<Book> listAllBooks() {
        return entityManager.createNamedQuery(ALL_BOOKS).getResultList();
    }

}
```

2.2.3.1.2. Writer Bean

The Writer Bean is also a Stateless Session Bean, with a Remote and a Local business interface. This bean provides write-like operations (that modify the database content). It is also secured to allow access only to a specific role.

Example 2.3. Stateless Session Bean: Writer business Interface

```
package org.ow2.jonas.examples.ear.writer;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;

/**
 * Remote interface for the bean Writer.
 * @author JOnAS team
 */
public interface Writer {

    /**
     * Persists a new {@link Author}.
     * @param author {@link Author} to add.
     */
    void addAuthor(final Author author);

    /**
     * Persists a new {@link Book}.
     * @param book {@link Book} to add.
     */
    void addBook(final Book book);

    /**
     * Cascade remove an {@link Author}.
     * @param author {@link Author} to be removed.
     */
    void removeAuthor(final Author author);

    /**
     * Cascade remove a {@link Book}.
     * @param book {@link Book} to be removed.
     */
    void removeBook(final Book book);
}
```

Through that interface, clients will be able to add new Authors and Books, but will also be able to remove them.

This bean uses (like the Reader) an `EntityManager` (injected through the `@PersistenceContext` annotation).

```
    /**
     * Entity manager used by this bean.
     */
    @PersistenceContext
    private EntityManager entityManager = null;
```

The EntityManager will be used to persists and destroy JPA entity instances.

```
    /**
     * Persists a new {@link Book}.
     * @param book {@link Book} to add.
     */
    public void addBook(final Book book) {
        entityManager.persist(book);
    }

    /**
     * Cascade remove a {@link Book}.
     * @param book {@link Book} to be removed.
     */
    public void removeBook(final Book book) {
        entityManager.remove(book);
    }
```

Security for the Writer bean is provided in a declarative way: the class is annotated with @DeclareRoles and @RolesAllowed.

- @DeclareRoles declares an array of role names that will be used later in the bean
- @RolesAllowed, when placed on the class (instead of a method) specifies the default set of roles that are allowed to call this bean's methods

In this sample, only the `earsample` role is authorized to actually use the Writer bean.

Here is the code of the Writer EJB:

```
package org.ow2.jonas.examples.ear.writer;

import javax.annotation.security.DeclareRoles;
import javax.annotation.security.RolesAllowed;
import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;

/**
 * This is an example of Session Bean, stateless, secured, available
 * with a Local and a Remote interface (with the same methods).
 * @author JOnAS team
 */
@Stateless
@Remote(RemoteWriter.class)
@Local(LocalWriter.class)
@DeclareRoles("earsample")
@RolesAllowed("earsample")
public class WriterBean implements LocalWriter, RemoteWriter {

    /**
     * Entity manager used by this bean.
     */
    @PersistenceContext
    private EntityManager entityManager = null;

    /**
     * Persists a new {@link Author}.
     * @param author {@link Author} to add.
     */
    public void addAuthor(final Author author) {
        entityManager.persist(author);
    }

    /**
     * Persists a new {@link Book}.
     * @param book {@link Book} to add.
     */
    public void addBook(final Book book) {
        entityManager.persist(book);
    }

    /**
     * Cascade remove an {@link Author}.
     * @param author {@link Author} to be removed.
     */
    public void removeAuthor(final Author author) {
        entityManager.remove(author);
    }

    /**
     * Cascade remove a {@link Book}.
     * @param book {@link Book} to be removed.
     */
    public void removeBook(final Book book) {
        entityManager.remove(book);
    }

}
```

2.2.3.1.3. Mailer Bean

The Mailer bean provides a business interface supporting status e-mail sending.

Example 2.4. Stateless Session Bean: Mailer Business Interface

```
package org.ow2.jonas.examples.ear.mail;

/**
 * The {@link Mailer} business interface is used to send a status
 * mail, with all the {@link org.ow2.jonas.examples.ear.entity.Author}s
 * and {@link org.ow2.jonas.examples.ear.entity.Book}s in the library.
 * @author Guillaume Sauthier
 */
public interface Mailer {

    /**
     * Send a mail to the given mail address.
     * @param address target mail address (must be of the form: xyz@abc.z)
     */
    void sendStatusMail(final String address);
}
```

This bean only provides a Remote interface. It is binded in the JNDI under the name specified in the mappedName attribute of the @Stateless annotation.

```
@Stateless(mappedName="myMailerBean")
@Remote(Mailer.class)
public class MailerBean implements Mailer {
```

The Mailer needs some other components to help it to execute it's job:

- A mail Session and a MimePartDataSource that will be used to create and send the e-mail
- A reference to the Reader bean

The mail Session and the MimePartDataSource are acquired though injection, using the @Resource annotation.

```
    /**
     * Mail Session used to send the Mail.
     */
    @Resource(mappedName="mailSession_1")
    private Session mailSession;

    /**
     * Template for the message's content.
     */
    @Resource(mappedName="mailMimePartDS_1")
    private MimePartDataSource mimePartDatasource;
```

The mappedName attribute of the @Resource specifies the JNDI name to be used when looking up theses components in the JNDI registry.

The reference to the Reader bean is injected with the help of the @EJB annotation. Notice that the annotation did not provide any JNDI or mapped name value.

```
    /**
     * {@link LocalReader} EJB (Local interface).
     */
    @EJB
    private LocalReader reader;
```

Here is the complete code of the Mailer bean:

```
package org.ow2.jonas.examples.ear.mail;

import java.util.Collection;
import java.util.Date;
import java.util.List;

import javax.annotation.Resource;
import javax.ejb.EJB;
import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.mail.Address;
import javax.mail.Message;
import javax.mail.MessageContext;
```

```
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimePartDataSource;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;
import org.ow2.jonas.examples.ear.reader.LocalReader;

/**
 * @author Guillaume Sauthier
 */
@Stateless(mappedName="myMailerBean")
@Remote(Mailer.class)
public class MailerBean implements Mailer {

    /**
     * Mail Session used to send the Mail.
     */
    @Resource(mappedName="mailSession_1")
    private Session mailSession;

    /**
     * Template for the message's content.
     */
    @Resource(mappedName="mailMimePartDS_1")
    private MimePartDataSource mimePartDatasource;

    /**
     * {@link LocalReader} EJB (Local interface).
     */
    @EJB
    private LocalReader reader;

    /**
     * Send a mail to the given mail address.
     * @param address target mail address (must be of the form: xyz@abc.z)
     * @see org.ow2.jonas.examples.ear.mail.Mailer#sendStatusMail(java.lang.String)
     */
    public void sendStatusMail(final String address) {

        Address mailAddress = null;
        try {
            mailAddress = new InternetAddress(address);
        } catch (AddressException e) {
            System.err.println("Invalid mail address: " + e.getMessage());
            e.printStackTrace();
            return;
        }

        MessageContext context = mimePartDatasource.getMessageContext();
        Message message = context.getMessage();
        try {
            message.setContent(getContent(), "text/plain");
        } catch (MessagingException e) {
            System.err.println("Cannot set message content:" + e.getMessage());
            e.printStackTrace(System.err);
            return;
        }

        Transport transport = null;
        try {
            transport = mailSession.getTransport(mailAddress);
        } catch (NoSuchProviderException e) {
            System.err.println("No provider found for @:" + address);
            e.printStackTrace(System.err);
            return;
        }
        try {
            transport.connect();
            transport.sendMessage(message, new Address[] {mailAddress});
            transport.close();
        } catch (MessagingException e) {
            System.err.println("Cannot send message:" + e.getMessage());
            e.printStackTrace(System.err);
            return;
        }

        System.out.println("Mail successfully sent to: " + address);
    }
```

```
    /**
     * Generate the mail's content.
     * @return the mail message content.
     */
    private String getContent() {

        StringBuilder sb = new StringBuilder();

        // Print Header
        sb.append("-------------------------------------------------\n");
        sb.append(" OW2 JOnAS EAR Sample Mailer Bean.\n");
        sb.append("-------------------------------------------------\n");
        sb.append("Generated the " + new Date() + "\n");
        sb.append("\n");

        // Print the Authors
        List<Author> authors = reader.listAllAuthors();
        sb.append("List of all registered Authors (" + authors.size() + ") and their Books:
\n");
        for (Author author : authors) {
            sb.append("  * " + author.getName() + " [" + author.getId() + "]\n");
            Collection<Book> books = author.getBooks();
            for (Book book : books) {
                sb.append("    - " + book.getTitle() + "[" + book.getId() + "]\n");
            }
        }

        // Print the Books
        sb.append("\n");
        sb.append("List of all registered Books:\n");
        List<Book> books = reader.listAllBooks();
        for (Book book : books) {
            sb.append("  * " + book.getTitle() + "[" + book.getAuthor().getName() + "]\n");
        }

        // Print the footer
        sb.append("\n");
        sb.append("Enjoy your new JOnAS !\n");
        sb.append("\n");
        sb.append("       -- JOnAS Team\n");

        return sb.toString();
    }

}
```

2.2.3.1.4. Initializer Bean

The Initializer bean is a kind of hidden bean: it does not provide operations that should be usable from the clients.

The purpose of this bean is to ensure that the JPA model is initialized with some default values. Subsequent calls to the JPA model will then always show JPA entities.

Since this session bean uses the Writer bean (required, because it writes values to the database), it needs to declare security.

Because this bean can be used by any client (secured or not), and it uses a secured bean, it will need to have to be executed under a given role. This role name is specified with the @RunAs annotation:

```
@RunAs("earsample")
public class InitializerBean implements Initializer {
```

Here is the complete code of the bean:

```
package org.ow2.jonas.examples.ear.init;

import javax.annotation.security.RunAs;
import javax.ejb.EJB;
import javax.ejb.Remote;
import javax.ejb.Stateless;

import org.ow2.jonas.examples.ear.entity.Author;
```

```
import org.ow2.jonas.examples.ear.entity.Book;
import org.ow2.jonas.examples.ear.reader.LocalReader;
import org.ow2.jonas.examples.ear.writer.LocalWriter;

/**
 * The {@link InitializerBean} EJB is here to initialize only once
 * the Database/Entities. It simply checks if there is some {@link Author}s
 * already persisted; if none are found, we will inject defaults values.
 * @author Guillaume Sauthier
 */
@Stateless(mappedName="myInitializerBean")
@Remote(Initializer.class)
@RunAs("earsample")
public class InitializerBean implements Initializer {

    /**
     * Injected reference to the {@link org.ow2.jonas.examples.ear.writer.Writer} EJB.
     */
    @EJB
    private LocalWriter writer;

    /**
     * Injected reference to the {@link org.ow2.jonas.examples.ear.reader.Reader} EJB.
     */
    @EJB
    private LocalReader reader;

    /**
     * Initialize the minimal set of entities needed by the sample.
     * @see org.ow2.jonas.examples.ear.init.Initializer#initializeEntities()
     */
    public void initializeEntities() {

        if (reader.findAuthor("Honore de Balzac") == null) {
            // Balzac was not persited, add it now.
            Author balzac = new Author("Honore de Balzac");
            Book pereGloriot = new Book("Le Pere Goriot", balzac);
            balzac.getBooks().add(pereGloriot);
            Book lesChouans = new Book("Les Chouans", balzac);
            balzac.getBooks().add(lesChouans);

            // Persists the Author and all of his books
            writer.addAuthor(balzac);
        }

        if (reader.findAuthor("Victor Hugo") == null) {
            // Hugo was not persited, add it now.
            Author hugo = new Author("Victor Hugo");
            hugo.addBook("Les Miserables");
            hugo.addBook("Notre-Dame de Paris");

            // Store
            writer.addAuthor(hugo);
        }
    }

}
```

2.2.3.2. MessageDriven Bean

The javaee5-earsample application provides a MessageDriven bean, aka an EJB that receives JMS messages. A MDB does not have a business interface, therefore, it is not exposed (through JNDI) to application clients or other EJBs. The only way to interact with them is to send messages to the JMS destination they're listening on.

A MDB is defined using the @MessageDriven annotation. The bean is also configured with the help of the @ActivationConfigProperty annotations. In this case, they define the JNDI name of the JMS destination (`"SampleQueue"`) and the destination type (can be `javax.jms.Queue` or `javax.jms.Topic` for JMS destinations). MDBs listening to JMS destinations need to implement the `javax.jms.MessageListener` interface.

```
@MessageDriven(activationConfig={
        @ActivationConfigProperty(propertyName="destination",
                                  propertyValue="SampleQueue"),
        @ActivationConfigProperty(propertyName="destinationType",
                                  propertyValue="javax.jms.Queue")
```

```
        })
@RunAs("earsample")
public class JMSMessageBean implements MessageListener {
```

This bean is also executed with the @RunAs annotation. This is needed because (like the Initializer) it uses the Writer bean, which is a secured bean.

Here is the code of the MDB:

```
package org.ow2.jonas.examples.ear.mdb;

import javax.annotation.security.RunAs;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;
import org.ow2.jonas.examples.ear.reader.LocalReader;
import org.ow2.jonas.examples.ear.writer.LocalWriter;

/**
 * The {@link JMSMessageBean} is a message driven bean activated when a JMS
 * {@link Message} comes to a given Destination.
 * For each new {@link Message}, this bean will create and persists a new
 * {@link Book} instance.
 * This MDB is annotated with {@link RunAs} because it uses a secured
 * business interface.
 * @author Guillaume Sauthier
 */
@MessageDriven(activationConfig={
        @ActivationConfigProperty(propertyName="destination",
                                  propertyValue="SampleQueue"),
        @ActivationConfigProperty(propertyName="destinationType",
                                  propertyValue="javax.jms.Queue")
        })
@RunAs("earsample")
public class JMSMessageBean implements MessageListener {

    /**
     * Secured business interface.
     */
    @EJB
    private LocalWriter writer;

    /**
     * Unsecured {@link LocalReader} business interface.
     */
    @EJB
    private LocalReader reader;

    /**
     * Called when a new JMS {@link Message} is received on the destination.
     * This method will use the {@link LocalWriter} Bean interface to add
     * Books to a given Author.
     * @param message {@link Message} containing {@link Book} title.
     * @see javax.jms.MessageListener#onMessage(javax.jms.Message)
     */
    public void onMessage(final Message message) {

        // TODO to be removed
        System.out.println("Received JMS Message: " + message);

        // Extract Message's text value
        String text = null;
        if (message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            try {
                text = textMessage.getText();
            } catch (JMSException e) {
                System.err.println("Unexpected Exception: " + e.getMessage());
                e.printStackTrace(System.err);
                return;
            }
        } else {
            // not a TextMessage, I don't know what to do with it
```

```
            return;
        }

        Author edition = reader.findAuthor("Editions XY");
        if (edition == null) {
            edition = new Author("Editions XY");
            writer.addAuthor(edition);
        }

        // Persists a new Book
        Book book = new Book(text, edition);
        writer.addBook(book);

    }

}
```

2.2.4. Web Tier: Servlets

Servlets are the web front end of the application, that's the presentation layer. Servlets are in charge of displaying business objects to the user and giving them handles to act on theses data objects.

Following is a screenshot obtained when pressing the "View Library Content" button on the home page:

The servlet `ExampleServlet` is used to generate the above web page.

As it is only a presentation front end, it does not manage the data itself. The servlet uses the Reader EJB to retrieve the content of the model.

The reference to the EJB is injected into the servlet in the Java EE 5 way: using annotations.

```
    /**
     * Link to the Local Reader bean. The bean will be injected by JOnAS.
     */
    @EJB
```

```
    private LocalReader readerBean;
```

Thousands of times easier than the old J2EE 1.4 way (InitialContext, lookup and narrow) !

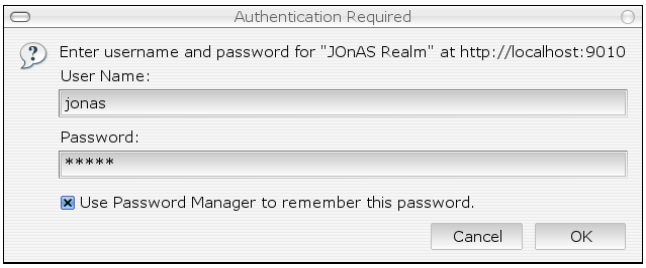
Here is the code of the read-only servlet:

```
package org.ow2.jonas.examples.ear.web;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collection;
import java.util.List;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;
import org.ow2.jonas.examples.ear.init.Initializer;
import org.ow2.jonas.examples.ear.reader.LocalReader;

/**
 * Defines a servlet that is accessing the two entities through a local session
 * bean.
 * @author Florent Benoit
 */
public class ExampleServlet extends HttpServlet {

    /**
     * Serializable class uid.
     */
    private static final long serialVersionUID = -3172627111841538912L;

    /**
     * Link to the Local Reader bean. Bean will be injected by JOnAS.
     */
    @EJB
    private LocalReader readerBean;

    /**
     * Link to the initializer bean.
     */
    @EJB
    private Initializer initializerBean;

    /**
     * Called by the server (via the service method) to allow a servlet to
     * handle a GET request.
     * @param request an HttpServletRequest object that contains the request the
     *        client has made of the servlet
     * @param response an HttpServletResponse object that contains the response
     *        the servlet sends to the client
     * @throws IOException if an input or output error is detected when the
     *         servlet handles the GET request
     * @throws ServletException if the request for the GET could not be handled
     */
    @Override
    public void doGet(final HttpServletRequest request, final HttpServletResponse response)
 throws IOException, ServletException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" \"http://
www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">");
        out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\" xml:lang=\"en\">");
        out.println("  <head>");
        out.println("    <link type=\"text/css\" href=\"ow2_jonas.css\" rel=\"stylesheet\"
 id=\"stylesheet\" />");
        out.println("    <title>Ear Sample of Servlet accessing an EJB</title>");
        out.println("  </head>");
        out.println("<body style=\"background : white; color : black;\">");

        out.println("  <div><a href=\"http://www.ow2.org\"><img src=\"img/logoOW2.png\" alt=
\"logo\"/></a></div>");
        out.println("  <div class=\"logos\">");
        out.println("    <img src=\"img/tomcat.gif\" alt=\"Tomcat Logo\"/>");
        out.println("    <img src=\"img/jetty.gif\" alt=\"Jetty Logo\"/>");
        out.println("    <img src=\"img/ow_jonas_logo.gif\" alt=\"JOnAS Logo\"/>");
```

```
        out.println("  </div>");

        out.println("  <div class=\"titlepage\">Ear sample of Servlet accessing an EJB</
div>");


        out.println("  <div class=\"links\">");
        initAuthorBooks(out);
        out.println("    <br />");
        out.println("  </div>");

        out.println("  <div class=\"links\">");
        displayAuthors(out);
        out.println("  </div>");

        out.println("  <div class=\"links\">");
        out.println("    <form action=\"secured/Admin\" method=\"get\">");
        out.println("      <div><input type=\"submit\" value=\"Modify Library Content\"/></
div>");
        out.println("    </form>");
        out.println("  </div>");

        out.println("  <div class=\"footer\">");
        out.println("    <p>");
        out.println("      <a href=\"http://validator.w3.org/check/referer\">");
        out.println("        <img src=\"img/valid-xhtml11.png\" alt=\"Valid XHTML 1.1!\"");
        out.println("             title=\"Valid XHTML 1.1!\" height=\"31\" width=\"88\" /
>");
        out.println("      </a>");
        out.println("      <a href=\"http://jigsaw.w3.org/css-validator/\">");
        out.println("        <img style=\"border:0;width:88px;height:31px\" src=\"img/
vcss.png\"");
        out.println("             title=\"Valid CSS!\" alt=\"Valid CSS!\" />");
        out.println("      </a>");
        out.println("    </p>");
        out.println("  </div>");

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

    /**
     * Init list of authors/books.
     * @param out the given writer
     */
    private void initAuthorBooks(final PrintWriter out) {
        out.println("Initialize authors and their books...<br/>");

        try {
            initializerBean.initializeEntities();
        } catch (Exception e) {
            displayException(out, "Cannot init list of authors with their books", e);
            return;
        }
    }

    /**
     * Display authors.
     * @param out the given writer
     */
    private void displayAuthors(final PrintWriter out) {
        out.println("Get authors");
        out.println("<br /><br />");

        // Get list of Authors
        List<Author> authors = null;
        try {
            authors = readerBean.listAllAuthors();
        } catch (Exception e) {
            displayException(out, "Cannot call listAllAuthors on the bean", e);
            return;
        }

        // List for each author, the name of books
        if (authors != null) {
            for (Author author : authors) {
                out.println("List of books with author '" + author.getName() + "' :");
                out.println("<ul>");
                Collection<Book> books = author.getBooks();
                if (books == null) {
                    out.println("<li>No book !</li>");
                } else {
```

```
                    for (Book book : books) {
                        out.println("<li>Title '" + book.getTitle() + "'.</li>");
                    }
                }
                out.println("</ul>");

            }
        } else {
            out.println("No author found !");
        }

    }

    /**
     * If there is an exception, print the exception.
     * @param out the given writer
     * @param errMsg the error message
     * @param e the content of the exception
     */
    private void displayException(final PrintWriter out, final String errMsg, final
 Exception e) {
        out.println("<p>Exception : " + errMsg);
        out.println("<pre>");
        e.printStackTrace(out);
        out.println("</pre></p>");
    }

}
```

The web application allows the user to change the model's content (ie add new Authors to the list of managed authors). This time, this is a write-like operation, so security is required to restrict unauthorized users from changing the model.



This behaviour is achieved through some additions into the `WEB-INF/web.xml`:

```
  <servlet>
    <servlet-name>AdminServlet</servlet-name>
    <servlet-class>org.ow2.jonas.examples.ear.web.AdminServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>AdminServlet</servlet-name>
    <url-pattern>/secured/*</url-pattern>
  </servlet-mapping>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <!-- Define the context-relative URL(s) to be protected -->
      <url-pattern>/secured/*</url-pattern>
      <!-- If you list http methods, only those methods are protected -->
      <http-method>DELETE</http-method>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
      <http-method>PUT</http-method>
    </web-resource-collection>
    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>earsample</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Default login configuration uses BASIC authentication -->
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>JOnAS Realm</realm-name>
  </login-config>
```

```
<!-- Security roles referenced by this web application -->
<security-role>
  <role-name>earsample</role-name>
</security-role>
```

By defining a `<security-constraint>` element, the web developer has decided to protect a given `<url-pattern>` (matching the servlet's mapping). Protection means that only authenticated users with the `earsample` `<role-name>` may access that page.

Then, once the user has authenticated himself (assuming he has the `earsample` role), he will be able to access the web page, allowing changes to the model:

2.2.5. Application Clients

The Java EE 5 EAR Sample provides multiple application clients (AC) showing how to interact with the application in different ways, and under different security levels:

- Not Secured AC: this client uses the Mailer and Reader bean, using the Reader, it displays the Authors and Books, and using the Mailer bean, it sends a mail with an equivalent content.
- JAAS Secured AC: this clients shows usage of JAAS authentication. It uses the Writer bean, which required a user with the "`earsample`" role, to insert new Authors and Books.
- JMS AC: this client mainly interacts with a dedicated JMS Queue to sends Book creation orders, then it waits some times and uses the Reader bean to see if the new Books were persisted.

2.2.5.1. Not Secured Application Client

This AC uses 3 beans that are freely available for anonymous usage: Initializer, Reader and Mailer.

The Client can be launched using the following command line:

```
>$ jclient -nowsgen $JONAS_BASE/deploy/javaee5-earsample.ear -jarClient not-secured-
application-client.jar
ClientContainer.info : Use the application client
'/tmp/client-deployer-coqp/EARDeployableImpl/javaee5-earsample.ear/not-secured-application-
client.jar'
of the Ear 'file:/tmp/client-deployer-coqp/EARDeployableImpl/javaee5-earsample.ear/'.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OW2 JOnAS :: EAR Sample :: Not Secured Application Client
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Initialization ... Done.
Get the RemoteReader Bean reference:
 org.ow2.jonas.examples.ear.reader.ReaderBean_org.ow2.jonas.examples.ear.reader.RemoteReader/7735352
List of authors, and their books:
 * Honore de Balzac
 -> Le Pere Goriot [id: 1]
 -> Les Chouans [id: 2]
 * Victor Hugo
 -> Les Miserables [id: 3]
 -> Notre-Dame de Paris [id: 4]
List of books:
 * Le Pere Goriot [Honore de Balzac]
 * Les Chouans [Honore de Balzac]
 * Les Miserables [Victor Hugo]
 * Notre-Dame de Paris [Victor Hugo]
Get the Mailer Bean reference:
 org.ow2.jonas.examples.ear.mail.MailerBean_org.ow2.jonas.examples.ear.mail.Mailer/7735352
Success.
```

The Initializer, Mailer and Reader beans are get via annotation based injection (no JNDI lookup is necessary).

```
@EJB
RemoteReader reader;
```

By default, a mail displaying the library content is sent to the `${user.name}@localhost` address. This value can be changed by adding another mail address into the command line:

```
>$ jclient -nowsgen $JONAS_BASE/deploy/javaee5-earsample.ear \
          -jarClient not-secured-application-client.jar someone@somewhere.org
```

Example 2.5. An example of e-mail

```
-------------------------------------------------
 OW2 JOnAS EAR Sample Mailer Bean.
-------------------------------------------------
Generated the Wed May 21 16:58:57 CEST 2008

List of all registered Authors (3) and their Books:
  * Honore de Balzac [1]
    - Le Pere Goriot[1]
    - Les Chouans[2]
  * Victor Hugo [2]
    - Les Miserables[3]
    - Notre-Dame de Paris[4]

List of all registered Books:
  * Le Pere Goriot[Honore de Balzac]
  * Les Chouans[Honore de Balzac]
  * Les Miserables[Victor Hugo]
  * Notre-Dame de Paris[Victor Hugo]

Enjoy your new JOnAS !

       -- JOnAS Team
```

Here is the not secured application client code:

```
package org.ow2.jonas.examples.ear.client;

import java.io.PrintStream;
import java.util.Collection;
import java.util.List;
```

```
import javax.ejb.EJB;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;
import org.ow2.jonas.examples.ear.init.Initializer;
import org.ow2.jonas.examples.ear.mail.Mailer;
import org.ow2.jonas.examples.ear.reader.RemoteReader;

/**
 * Simple Application Client.
 * @author Guillaume Sauthier
 */
public final class NotSecuredApplicationClient {

    /**
     * Empty default constructor for utility class.
     */
    private NotSecuredApplicationClient() {

    }

    /**
      * Link to the initializer bean.
      */
     @EJB
     static private Initializer initializerBean;

    /**
      * Link to the Remote Reader bean. Bean will be injected by JOnAS.
      */
     @EJB
     static private RemoteReader readerBean;

     /**
      * Link to the Mailer bean. Bean will be injected by JOnAS.
      */
     @EJB
     static private Mailer mailerBean;


    /**
     * @param args Command line arguments
     *
     */
    public static void main(final String[] args)  {

        PrintStream out = System.out;

        // Print Header
        out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
        out.println("OW2 JOnAS :: EAR Sample :: Not Secured Application Client  ");
        out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

        // Init datas if needed
        out.print("Initialization ... ");
        initializerBean.initializeEntities();
        out.println("Done.");


        out.println("The RemoteReader Bean reference is : " + readerBean);

        // List registered authors and their books
        out.println("List of authors, and their books:");
        // retrieve again the authors list (now it has been initialized)
        List<Author> authors = readerBean.listAllAuthors();
        for (Author author : authors) {
            out.println(" * " + author.getName());
            Collection<Book> books = author.getBooks();
            for (Book book : books) {
                out.println(" -> " + book.getTitle() + " [id: " + book.getId() + "]");
            }
        }

        // Only list registered books
        List<Book> books = readerBean.listAllBooks();
        out.println("List of books:");
        for(Book book : books) {
            out.println(" * " + book.getTitle() + " [" + book.getAuthor().getName() + "]");
        }

        // Use the Mailer bean to send
        // Use the first command line argument as a mail address.
        // Fall back to a reasonable default ${user.name}@localhost
```

```
        String address = System.getProperty("user.name") + "@localhost";
        if (args.length > 0) {
            // Got an argument, use it ...
            address = args[0];
        }

        out.println("The Mailer Bean reference is : " + mailerBean);

        // Call the Mailer bean to send the expected e-mail.
        mailerBean.sendStatusMail(address);

        // OK, we're done
        out.println("Success.");

    }
}
```

2.2.5.2. JAAS Secured Application Client

This AC is secured using JAAS. It means that the client will be authenticated at startup.

The Client can be launched using the following command line:

```
>$ jclient -nowsgen $JONAS_BASE/deploy/javaee5-earsample.ear -jarClient jaas-secured-
application-client.jar
ClientContainer.info : Use the application client
'/tmp/client-deployer-sauthieg/EARDeployableImpl/javaee5-earsample.ear/jaas-secured-
application-client.jar'
of the Ear 'file:/tmp/client-deployer-sauthieg/EARDeployableImpl/javaee5-earsample.ear/'.
ClientContainer.info : Using the login/password specified in the jonas-client.xml file with
 a specific CallbackHandler
ClientContainer.info : Using JAAS loginContext 'javaee5-earsample'
from the file 'jar:file:/tmp/client-deployer-sauthieg/EARDeployableImpl/javaee5-
earsample.ear/jaas-secured-application-client.jar!/jaas.config'.
ClientContainer.info : Starting client...
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OW2 JOnAS :: EAR Sample :: Secured Application Client
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Initialization ... Done.
Get the RemoteReader Bean reference:
 org.ow2.jonas.examples.ear.reader.ReaderBean_org.ow2.jonas.examples.ear.reader.RemoteReader/7735352
List of authors, and their books:
 * Honore de Balzac
 -> Le Pere Goriot [id: 1]
 -> Les Chouans [id: 2]
 * Victor Hugo
 -> Les Miserables [id: 3]
 -> Notre-Dame de Paris [id: 4]
Get the RemoteWriter Bean reference:
 org.ow2.jonas.examples.ear.writer.WriterBean_org.ow2.jonas.examples.ear.writer.RemoteWriter/7735352
Created a new Author:
org.ow2.jonas.examples.ear.entity.Author[id=0, name=Emile Zola]
Updated authors' list:
 * Honore de Balzac
 -> Le Pere Goriot [id: 1]
 -> Les Chouans [id: 2]
 * Victor Hugo
 -> Les Miserables [id: 3]
 -> Notre-Dame de Paris [id: 4]
 * Emile Zola
 -> Germinal [id: 45]
 -> La Bete Humaine [id: 46]
Cleaned added Author.
Success.
```

This AC has been configured to use the LoginCallbackHandler. The CallbackHandler will be configured by JOnAS to use the jonas/jonas username and password pair. So, nothing has to be done to provide a password interactively.

Example 2.6. META-INF/application-client.xml

```
<application-client
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                      http://java.sun.com/xml/ns/javaee/application-client_5.xsd"
  version="5">
  <display-name>OW2 JOnAS :: EAR Sample :: Secured Application Client</display-name>
  <callback-handler>org.ow2.jonas.security.auth.callback.LoginCallbackHandler</callback-
handler>
</application-client>
```

Example 2.7. META-INF/jonas-client.xml

```
<jonas-client 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://jonas.ow2.org/ns/jonas-client_5_0.xsd" >

  <jonas-security>
    <jaasfile>jaas.config</jaasfile>
    <jaasentry>javaee5-earsample</jaasentry>
    <username>jonas</username>
    <password>jonas</password>
  </jonas-security>

</jonas-client>
```

Example 2.8. jaas.config

```
javaee5-earsample {
    // Login Module to use for the example javaee5-earsample.

    // First, use a LoginModule for the authentication
    // Use the resource memrlm_1
    org.ow2.jonas.security.auth.spi.JResourceLoginModule required
    resourceName="memrlm_1"
    ;

    // Use the login module to propagate security to the JOnAS server
    org.ow2.jonas.security.auth.spi.ClientLoginModule  required
    ;
};
```

The `CallbackHandler` is configured using the `jonas-security` element: it will load the specified JAAS configuration file (`jaas.config`), then load the entry named `javaee5-earsample` and will create a dedicated `LoginModule` chain. The ClientContainer will then configure the `CallbackHandler` to provide a specified username and password when requested by the LoginModules.

As the authentication is done by the ClientContainer at startup, there are no security concerns in the code. The ClientContainer guarantees that if this code is executed, it is run by an authenticated user. If the user could not authenticate, the ClientContainer will exit before running the real client's code.

```
package org.ow2.jonas.examples.ear.client;

import java.io.PrintStream;
import java.util.Collection;
import java.util.List;
import java.text.MessageFormat;

import javax.ejb.EJB;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;
import org.ow2.jonas.examples.ear.init.Initializer;
import org.ow2.jonas.examples.ear.reader.RemoteReader;
import org.ow2.jonas.examples.ear.writer.RemoteWriter;

/**
 * Simple Application Client.
 * @author Guillaume Sauthier
```

```
 */
public final class SecuredApplicationClient {

    /**
     * Empty default constructor for utility class.
     */
    private SecuredApplicationClient() {

    }

       /**
      * Link to the initializer bean.
      */
     @EJB
     static private Initializer initializerBean;

    /**
      * Link to the Remote Reader bean. Bean will be injected by JOnAS.
      */
     @EJB
     static private RemoteReader readerBean;

     /**
      * Link to the Remote Writer bean. Bean will be injected by JOnAS.
      */
     @EJB
     static private RemoteWriter writerBean;

    /**
     * @param args Command line arguments
     *
     */
    public static void main(final String[] args) {

        PrintStream out = System.out;

        // Print Header
        out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
        out.println("OW2 JOnAS :: EAR Sample :: Secured Application Client      ");
        out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
        // Init. data if needed
        out.print("Initialization ... ");
        initializerBean.initializeEntities();
        out.println("Done.");

        out.println("The RemoteReader Bean reference is: " + readerBean);

        // List registered authors and their books
        out.println("List of authors, and their books:");
        // retrieve again the authors list (now it has been initialized)
        List<Author> authors = readerBean.listAllAuthors();
        for (Author author : authors) {
            out.println(" * " + author.getName());
            Collection<Book> books = author.getBooks();
            for (Book book : books) {
                out.println(MessageFormat.format(" -> {0} [id: {1}]", book.getTitle(),
 book.getId()));
            }
        }

        // Use the secured Bean (RemoteWriter)
        // ===================================
        out.println("Get the RemoteWriter Bean reference: " + writerBean);

        // Add another author, and some books
        Author zola = new Author("Emile Zola");
        Book germinal = new Book("Germinal", zola);
        Book beteHumaine = new Book("La Bete Humaine", zola);
        zola.getBooks().add(germinal);
        zola.getBooks().add(beteHumaine);

        // Display Author before storage
        out.println("Created a new Author: ");
        out.println(zola.toString());

        // Persists
        writerBean.addAuthor(zola);

        // See the new content
        out.println("Updated authors' list:");
        authors = readerBean.listAllAuthors();
        for (Author author : authors) {
            out.println(" * " + author.getName());
```

```
            Collection<Book> books = author.getBooks();
            for (Book book : books) {
                out.println(MessageFormat.format(" -> {0} [id: {1}]", book.getTitle(),
 book.getId()));
            }
        }

        // Remove Zola (and its books), so that next time the client
        // is executed, we can add them again.
        writerBean.removeAuthor(zola);
        out.println("Cleaned added Author.");
        out.println("Success.");

    }
}
```

2.2.5.3. JMS Application Client

This AC demonstrates how to connect a client to the application using JMS instead of using RMI EJB objects.

The Client can be launched using the following command line:

```
>$ jclient -nowsgen $JONAS_BASE/deploy/javaee5-earsample.ear
ClientContainer.warn : There are 3 clients in this ear, choosing the first one : jms-
application-client.jar
ClientContainer.info : Use the application client '/tmp/client-deployer-sauthieg/
EARDeployableImpl/javaee5-earsample.ear/jms-application-client.jar'
of the Ear 'file:/tmp/client-deployer-sauthieg/EARDeployableImpl/javaee5-earsample.ear/'.
ClientContainer.info : Starting client...
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OW2 JOnAS :: EAR Sample :: Messager Application Client
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Initialization ... Done.
Sended creation order for 'Encyclopedia Universalis Vol.0'
Sended creation order for 'Encyclopedia Universalis Vol.1'
Sended creation order for 'Encyclopedia Universalis Vol.2'
Sended creation order for 'Encyclopedia Universalis Vol.3'
Sended creation order for 'Encyclopedia Universalis Vol.4'
Sended creation order for 'Encyclopedia Universalis Vol.5'
Sended creation order for 'Encyclopedia Universalis Vol.6'
Sended creation order for 'Encyclopedia Universalis Vol.7'
Sended creation order for 'Encyclopedia Universalis Vol.8'
Sended creation order for 'Encyclopedia Universalis Vol.9'
Wait for 2500 ms...
Get Reader Bean ...
 * Honore de Balzac
 -> Le Pere Goriot [id: 1]
 -> Les Chouans [id: 2]
 * Victor Hugo
 -> Les Miserables [id: 3]
 -> Notre-Dame de Paris [id: 4]
 * Editions XY
 -> Encyclopedia Universalis Vol.0 [id: 5]
 -> Encyclopedia Universalis Vol.1 [id: 6]
 -> Encyclopedia Universalis Vol.2 [id: 7]
 -> Encyclopedia Universalis Vol.3 [id: 8]
 -> Encyclopedia Universalis Vol.4 [id: 9]
 -> Encyclopedia Universalis Vol.5 [id: 10]
 -> Encyclopedia Universalis Vol.6 [id: 11]
 -> Encyclopedia Universalis Vol.7 [id: 12]
 -> Encyclopedia Universalis Vol.8 [id: 13]
 -> Encyclopedia Universalis Vol.9 [id: 14]
Success.
```

The client acts as a message producer: it retrieves a JMS `Queue` object (named `SampleQueue`) and a JMS `QueueConnectionFactory`. Then, it uses both to send asynchronous messages to the server.

Each message represents a book registration order and contains the name of the Book to create.

On the application's side (in other words, on the server), there is a dedicated MessageDrivenBean (see the Section 2.2.3.2, “MessageDriven Bean”), that will be invoked for all the JMS messages received by the `Queue`.

Usage of JMS objects have to be declared in standard and specific deployment descriptors (using `resource-ref` and/or `resource-env-ref`, respectively `jonas-resource` and/or `jonas-resource-env`)

This JMS example shows how to use resource injection via @Resource annotation (see JMS client code below).

Example 2.9. META-INF/application-client.xml

```
<application-client
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                      http://java.sun.com/xml/ns/javaee/application-client_5.xsd"
  version="5">

  <display-name>OW2 JOnAS :: EAR Sample :: JMS Application Client</display-name>

  <!-- The JMS ConnectionFactory to use -->
  <resource-ref>
    <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
    <res-type>javax.jms.QueueConnectionFactory</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>

  <!-- The JMS Queue where Messages will be send -->
  <resource-env-ref>
    <resource-env-ref-name>jms/SampleQueue</resource-env-ref-name>
    <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
  </resource-env-ref>

</application-client>
```

Example 2.10. META-INF/jonas-client.xml

```
<jonas-client 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://jonas.ow2.org/ns/jonas-client_5_0.xsd" >
  <jonas-resource>
    <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
    <jndi-name>JQCF</jndi-name>
  </jonas-resource>

  <jonas-resource-env>
    <resource-env-ref-name>jms/SampleQueue</resource-env-ref-name>
    <jndi-name>SampleQueue</jndi-name>
  </jonas-resource-env>
</jonas-client>
```

Here is the code of the JMS client:

```
package org.ow2.jonas.examples.ear.client;

import java.io.PrintStream;
import java.util.Collection;
import java.util.List;
import java.text.MessageFormat;

import javax.jms.*;
import javax.ejb.EJB;
import javax.annotation.Resource;

import org.ow2.jonas.examples.ear.entity.Author;
import org.ow2.jonas.examples.ear.entity.Book;
import org.ow2.jonas.examples.ear.init.Initializer;
import org.ow2.jonas.examples.ear.reader.RemoteReader;

/**
 * This application-client shows usage of JMS destinations to
 * interact with the server-side application.
 * @author Guillaume Sauthier
 */
public final class JMSApplicationClient {

    /**
```

```
     * Number of Books to be created.
     */
    private static final int ITERATION_NUMBER = 10;

    /**
      * Link to the initializer bean.
      */
     @EJB
     private static  Initializer initializerBean;

    /**
     *  JMS conectionFactoery
     */
    // Resource injection
     @Resource(mappedName="JQCF")
      private static ConnectionFactory factory;

    /**
     *   JMS Queue  SampleQueue
     */
    // Resource injection
     @Resource(mappedName="SampleQueue")
     private static Queue queue;


    /**
      * Link to the Remote Reader bean. Bean will be injected by JOnAS.
      */
     @EJB
     static private RemoteReader readerBean;

    /**
     * Empty default constructor for utility class.
     */
    private JMSApplicationClient() {

    }

    /**
     * @param args Command line arguments
     * @throws Exception JMS
     */
    public static void main(final String[] args) throws Exception {

        PrintStream out = System.out;

        // Print Header
        out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
        out.println("OW2 JOnAS :: EAR Sample :: Messager Application Client     ");
        out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");




        // Init. data if needed
        out.print("Initialization ... ");

        initializerBean.initializeEntities();
        out.println("Done.");

        // Send Book creation Messages
        Connection connection = factory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer= session.createProducer(queue);

        for (int i = 0; i < ITERATION_NUMBER; i++) {
            String title = "Encyclopedia Universalis Vol." + i;
            Message message = session.createTextMessage(title);
            messageProducer.send(message);
            out.println("Sended creation order for '" + title + "'");
        }

        // Close JMS objects
        messageProducer.close();
        session.close();
        connection.close();

        // Wait for some time ...
        // Remember JMS is for asynchronous messages :)
        final long period = 2500;
        out.println(MessageFormat.format("Wait for {0} ms...", period));
        Thread.sleep(period);
```

```
        out.println(MessageFormat.format("The RemoteReader Bean reference is: {0}",
 readerBean));
        // List Authors and Books
        List<Author> authors = readerBean.listAllAuthors();
        for (Author author : authors) {
            out.println(MessageFormat.format(" * {0}", author.getName()));
            Collection<Book> books = author.getBooks();
            for (Book book : books) {
                out.println(MessageFormat.format(" -> {0} [id: {1}]", book.getTitle(),
 book.getId()));
            }
        }

        out.println("Success.");

    }
}
```

2.2.6. EAR

The `javaee5-earsample.ear` file is the enterprise archive wrapping all the Java EE modules into only one deployment unit.

The EAR is all about packaging: there is no code and no resources (eg, classes, ...) directly inside of the .ear.

Here is an overview of the ear structure:

```
javaee5-earsample.ear:
  META-INF/application.xml              1
  ejb3.jar                              2
  javaee5-earsample.war                 3
  jms-application-client.jar            4
  not-secured-application-client.jar    5
  jaas-secured-application-client.jar   6
```

1. Optional, inner jars can be discovered with .ear introspection
2. The jar file containing the EJB3
3. The Web Application archive
4. The Application Client using JMS to interact with the application
5. The Application Client using the Reader EJB without security
6. The Application Client using the Reader and Writer EJBs after being authenticated with JAAS

Appendix A. Download and installation instructions

A.1. Where can 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.

The JOnAS project is developped using SVN. All information for getting or browsing the 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] offers links to easily download JOnAS.

A.3. How can I install JOnAS?

The JOnAS distribution can be downloaded as a .tar.gz or .zip file.

The installation process simply consists of unzipping the downloaded .tar.gz file.

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

- Unix platforms

```
bash> tar -zxvf <jonas-file-name>.tar.gz
 jonas-full-5.x.y/
 jonas-full-5.x.y/conf/
 jonas-full-5.x.y/examples/
 jonas-full-5.x.y/examples/javaee5-earsample/
 jonas-full-5.x.y/examples/javaee5-earsample/etc/
...
```

Caution

Be aware that if the same version of JOnAS has already been unpacked in the same directory, 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

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

A.4. Pre-requisites

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

- a J2SE SDK 1.5 Java virtual machine

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

The most commonly used is SUN's (Java 2 Platform, Standard Edition [http://java.sun.com/javase/downloads/index_jdk5.jsp]), but there are others, such as BEA JRockit [http://dev2dev.bea.com/jrockit/], IBM developper kits [http://www-128.ibm.com/developerworks/java/jdk/] or other free/open source certified implementations.

- Ant 1.7 and BCEL

  The binary version of Ant 1.7 must be downloaded from the Apache Ant web site [http://ant.apache.org/bindownload.cgi] and installed

```
bash> tar -jxvf apache-ant-1.7.0-bin.tar.bz2
or
bash> unzip apache-ant-1.7.0-bin.zip
```

  Set the ANT_HOME environment variable and update the path:

```
on Unix platforms:
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 Apache Jakarta web site [http://www.apache.org/dist/jakarta/bcel/binaries/]. The bcel-5.1.jar must be installed in the `$ANT_HOME/lib/` directory.

Appendix B. JOnAS 5 distribution description

Here we describe the directory tree you get under your installation directory (`JONAS_ROOT` environment).

B.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 Chapter 2, *Learning JOnAS by examples*

- the `lib/` directory [1]

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

Directory	Description
`bootstrap/`	Jars loaded by the JOnAS bootstrap

Directory	Description
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.

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 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.