Enhydra Application Architecture

Tanja Jovanovic

Table of Contents

List of Examples

Chapter 1. Introduction

This document describes Enhydra applications. It describes the architecture of the classes you must write to create an Enhydra application.

After the Enhydra Web application is created, it is compiled. After compiling, the result is a single jar file that is a Servlet. This Servlet may be served to the net by different servlet continers (Tomcat, Jetty,...), or it may be installed in any Servlet-capable Web server.

The Enhydra Application must consist of one application object, and a set of presentation objects (one for each URL in the application).

Chapter 2. The Application Object

The application object is the central hub of your application. It is responsible for holding all application-wide data needed by the framework. Some examples of this data are:

- the current running status (started/stopped/dead)
- the config file used to initialize the application
- the name of the application
- the log channel to use for logging
- the pointers to parts of the framework (session manager, database manager)
- preprocessor function used on all requests.

Enhydra provides a class, StandardApplication [../Modules/EAF/user-doc-api/com/lutris/appserver/server/StandardApplication.html], that does all this work. Only the pieces needed for the application must be overridden.

For example, one of Enhydra example applications is *Disc Rack* application. The file discRack/DiscRack.java is application's *Application object*. It extends *StandardApplication*, and extends the preprocessor function (just to show it's there).

Application typically does three things to the application object:

- reads settings from the config file at startup
- extends the preprocessor (to check for HTTP basic authorization, for example)
- stores application-wide data structures.

As an example, suppose your application has a dozen URLs (pages), but they all share a single hashtable or vector of some kind of data. You could make this a field of your application object and all the pages can easily access it. See also business objects, below.

Aside from the preprocessor function, which is optional, application objects do not deal with HTML, handle requests or otherwise talk to the net. This is the job of the presentation objects.

Note: the application object must implement the interface com.lutris.appserver.server.Application [../Modules/EAF/user-doc-api/com/lutris/appserver/server/Application.html]. Also, it is recommend that it extends the class com.lutris.appserver.server.StandardApplication [../Modules/EAF/user-doc-api/com/lutris/appserver/server/StandardApplication.html].

In the application's config file, the key:

- *Server.AppClass* if config file is <appName.conf> or
- *Server/AppClass* if config file is web.xml

tells the framework which class to use as the application object.

For example, for DiscRack example, in web.xml the value of this key may be:

```
<env-entry>
  <env-entry-name>Server/AppClass</env-entry-name>
  <env-entry-value>discRack.DiscRack</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Most Web-based applications consist of three fundamental types of components: presentation objects, business objects and data objects.

Chapter 3. The Presentation Object

Presentation Object contains the logic that presents information to an external source and obtains input from that source. The presentation logic generally provides menus of options to allow the user to navigate through the different parts of the application, and it manipulates the input and output fields on the display device. Frequently the presentation component also performs a limited amount of input data validation.

Presentation object must be provided for each URL (page) in the application. The presentation object is given the request in object form and is responsible for servicing the request. It is also given a response object to use to write data in response (very similar to the Servlet API's service() method). Almost all presentation objects handle GET requests and respond by writing HTML to the Net, but they may also, for example, read in files sent by a POST request, or send Java serialized objects to an Applet.

Presentation objects were given that name because they are in charge of how the application's data and results are presented to the client (the Web browser). See the section "Recommends" below for more details.

When a request is sent to the application, the framework receives it (through the Servlet interface), and determines which presentation object to pass the request on to. The presentation object is then located (via the class loader), instantiated and called. All this happens automatically. Presentation objects do not need to notify the framework that they exist. The framework, driven by the URLs it is asked for, will attempt to locate the corresponding Java classes.

Only requests for URLs that end in ".po" (presentation object) will result in calling a presentation object. All other types of requests are handled by serving a static file, just like a normal Web server. This lets you mix dynamic content pages (.po) in with normal html files (.html) and images (.gif, .jpeg). Rather than being files on a disk, these are typically archived into the application's jar file. The class loader is used to read the files (getResourceAsStream()). This lets you archive your entire application, pictures and all, into a single jar file.

The Enhydra application has two presentation objects: the URLs Welcome.po and Redirect.po. They are the only URLs the application responds to. All other URLs result in the standard file not found response (an optional special case is the URL /).

In the application's config file, the key :

- *Server.PresentationPrefix* if config file is <appName.conf> or
- *Server/PresentationPrefix* if config file is web.xml

the root of the Java packages containing the presentation objects (with "/" separators). Suppose that the presentation prefix is "application/foo/bar". Then suppose a request arrives for the URL "login/remote/Login.po". The framework will put these two together resulting in the class name "application.foo.bar.login.remote.Login". The framework will then attempt to load this class. If it is found, the request is sent to it. If it is not found a file, not found response is returned.

For example, for DiscRack example, in web.xml the value of this key may be:

```
<env-entry>
  <env-entry-name>Server/PresentationPrefix</env-entry-name>
  <env-entry-value>discRack/presentation</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Note: presentation objects must implement the interface com.lutris.appserver.server.httpPresentation.HttpPresentation [../Modules/EAF/user-doc-api/com/lutris/appserver/server/httpPresentation/HttpPresentation.html].

Because most presentation objects emit dynamic HTML, they can be written using Document Object Model (DOM) access to the HTML (see Writing Presentation Objects with XMLC below). They can also be written by hand, simply by writing a Java class that implements *HttpPresentation*.

Chapter 4. Writing Presentation Objects with XMLC

The XML Compiler (XMLC) reads normal HTML files and creates Java classes that contain and represent the exact same HTML content. The Document Object Model (DOM) is used to provide access methods to read and modify the content. You would then write a "by hand" presentation object which uses the XMLC generated classes as a library.

In the HTML, you add

```
ID=Name
```

to whatever tag you want access to.

Example 4.1. Examples of using ID attribute

```
<B ID=FirstName>John</B>
<I ID=LastName>Doe</I>
```

The resulting class will have *getFirstName()*, *setFirstName()*, *getLastName()* and *setLastName()* classes. You simply call the set method, then call the class's toHtml() method to get the HTML for the entire page (which you would then write out to the Net). If you want DOM access to the whole page, simply add an ID field to the <BODY> tag.

We have found that for large projects is it very important to minimize the interference between the software engineers and the graphic designers, who use HTML tools that have a tendency to reformat entire files. The small amount of extra work to coordinate with designers in the beginning of the project will be paid back many times over at the end of the project when the engineers can fix bugs while the graphic artists redo the entire style of the site, and the two changes will not interfere with each other.

DOM allows access to XML files. So the XML Compiler will also allow for presentation objects to serve dynamic content XML.

Chapter 5. Writing Presentation Objects Using Java

At times you will have a presentation object that does not emit HTML, or needs to do a lot of computation in Java. An example of this is if you have a login screen (with an HTML form) that sends the results to a presentation object LoginProcessor.po. This presentation object examines the username and password and decides if the user is allowed to log in. If they are, the user's session object is marked as logged in, and an HTTP redirect to Main.po is returned. If they aren't, an HTTP redirect back to Login.po is returned. So either way the presentation object does not need to handle any HTML. In this case you would simply write a Java class to do the work. By providing a *run()* method, it meets the requirements to implement *HttpPresentation*.

A good rule of thumb is: if you have a lot more code than content, you may want to simply write the class yourself.

Chapter 6. The Business Object

Business Objects contain the application logic that governs the business function and process. Business objects are invoked either by a presentation component when a user requests an option, or by another business function.

The business functions generally perform some type of data manipulation.

Chapter 7. The Data Object

Data Objects contain the logic that interfaces with a data storage system, such as database systems or hierarchical file systems, or with some other type of external data source such as a data feed or an external application system.

Business objects invoke Data Objects to save persistent state.

Chapter 8. Recommends

Through years of experience, we strongly recommend that you break your application up into three categories of objects: presentation objects, business objects, and data objects. Presentation objects handle how the application data is presented to the clients (Web browsers). Any and all HTML is kept in the presentation objects. Data objects get and set the data your application manipulates, from a file, a database or even a hard-coded list. All database code, or file reading code, is kept in the data objects. Business objects handle all the "business logic". All the policy decisions, algorithms and data manipulation. Essentially everything left over after you quarantine all the HTML (and HTTP) to the presentation objects, and quarantine all the database/file access code to the data objects.

Designing your application this way minimizes the impact on your application when you switch databases, file formats or URL layouts. The Enhydra Application Framework only requires that you use presentation objects. The business and data classes are optional.

An additional benefit of designing your application this way is that you will be able to use the Data Object Design Studio (DODS) (html [../Tools/DODS/index_dods.html] , pdf [../Tools/DODS/index_dods.pdf]) to create your data objects for you. You graphically create your objects, and it creates both Java code that implements them (with get and set methods that talk SQL to the database), and the SQL code to create the required tables in your database.

Chapter 9. Presentation Manager

The presentation manager handles the loading and execution of presentation objects in the context of an Enhydra application. The presentation manager transforms the name of the presentation object into a URL, uses the specified class loader to load the presentation object, and then executes the presentation object by executing its run() method. There is one instance of a presentation manager per Enhydra application. The presentation manager can also cache presentation objects in memory and any associated files that are part of the application.

It uses the Enhydra class loader to get these other files.

The presentation manager is contained within an instance of HttpPresentationServlet [../Modules/EAF/user-doc-implementation/com/lutris/appserver/server/httpPresentation/servlet/HttpPresentationServlet.html] class. This design allows a servlet to have multiple presentation objects managed by the presentation manager. The presentation manager also manages the resources necessary to execute the presentation objects.

The presentation manager also provides the key with which the session manager uses to locate a session. The key either is a cookie or generated by URL rewriting.

Chapter 10. The Session Manager

The SessionManager [../Modules/EAF/user-doc-api/com/lutris/appserver/server/session/SessionManager.html] is one part of the Enhydra Application Framework. The framework automatically issues users a cookie and creates a Session [../Modules/EAF/user-doc-api/com/lutris/appserver/server/session/Session.html] object. The cookie is a secure, opaque identifier. All the application data is kept safely inside the application. Every request the user makes, when it is given to the presentation objects, contains the user's session object. Inside the session object is a SessionData [../Modules/EAF/user-doc-implementation/com/lutris/appserver/server/session/SessionData.html] object. This is a totally flexible container for whatever the application wants to store.

The session manager and session objects provide a flexible, lightweight mechanism that enables stateful programming on the Web. Enhydra provides a general implementation of session management that serves as a basis for more sophisticated state models (a session is a series of requests from the same user that occur during a time-period). Session management gives servlets and other server-side applications the ability to keep state about a user as the user moves through the application. Enhydra maintains user state by creating a Session object for each user. These Session objects are stored and maintained on the server. When a user first makes a request to an application, the user the session manager assigns a new Session object and a unique session ID. The session ID matches the user with the Session object in subsequent requests. The Session object is then passed as part of the request to the servlets that handle the request. Servlets can add information to Session objects or read information from them. After the user has been idle for more than a certain period, the user's session becomes invalid and the Session Manager destroys the corresponding Session object.

Since Enhydra 6.0, different implementations of the session manager have been developed. The class that implements the one used with each of the applications is to be stated in the configuration file, for example:

```
SessionManager.Class=com.lutris.appserver.server.sessionEnhydra.SimpleServletSessionManager
```

If the *SessionManager.Class* is not specified, the old *com.lutris.appserver.server.sessionEnhydra.StandardSessionManager* is used (it is kept for compatibility reasons, but it's use is not encouraged).

The available session manager adapters are:

- *com.lutris.appserver.server.sessionEnhydra.SimpleServletSessionManager* - this is a simple session manager which interconnects the servlet container sessions and enhydra sessions by using the same session keys (generated by session container).
- *com.lutris.appserver.server.sessionContainerAdapter.ContainerAdapterSessionManager* - simple session manager to be used with servlet container capable of managing their sessions. It uses HttpSession to keep the session data. The sessions are completely managed by the session container and are configured in the servlet container configuration files. Any session configuration parameters defined in the application configuration file are ignored (except sessionManager.Class). The persistence across restarts of the application and container is relised by the appropriate servlet container mechanisms.
- *com.lutris.appserver.server.sessionContainerAdapter.JmxContainerAdapterSessionManager* - Tomcat specific session manager, extends ContainerAdapterSessionManager, by using JMX MBeans to obtain some session information from the session container.

NOTE: For the latter two containers, session data must be serializable in order to fully utilize the persistence obtained by the session container.

Any data that is application-wide (shared across all pages and all users) should be kept in the application object. Any data that is user-wide (needed across all URLs a user goes to, but one copy per user) should be kept in SessionData. Any data that is page-specific (only needed by one page, but shared across all accesses to the page) should be stored as static fields of the presentation object.

Chapter 11. The Database Manager

The Database Manager is another (optional) part of the Enhydra Application Framework. An application may create an instance of the StandardDatabaseManager [../Tools/DODS/user-doc/com/lutris/appserver/server/sql/StandardDatabaseManager.html] class by specifying keys in the application config file.

Since Enhydra 6.0, in application's configuration file is added parameter that indicates which database manager implementation is used. The parameter is *DatabaseManager.Class*. For example:

```
DatabaseManager.Class = com.lutris.appserver.server.sql.StandardDatabaseManager
```

At the moment, *com.lutris.appserver.server.sql.StandardDatabaseManager* is the only database manager implementation. If the *DatabaseManager.Class* is not specified, *com.lutris.appserver.server.sql.StandardDatabaseManager* implementation is used.

The database is the component that manages a pool of connections across any number of logical databases, allowing for much faster database access. SQL queries are also cached, again allowing for faster database access.

Logical databases hide the nuances across different JDBC and database implementation and contain a set of connections. These connections are shared across thread boundaries. The connections are responsible for maintaining a connection to a JDBC driver and the state of a database connection, including the current statement(s) and result set(s) that are in progress.

Chapter 12. The Administration Manager

For managing Enhydra applications are used JOnAS Administration web application and Enhydra Admin application.

These applications are graphical tool that allow a system manager to configure and monitor an instance of Enhydra and associated applications. All configuration information for Enhydra and Enhydra applications are stored in configuration files. When Enhydra starts, it reads these configuration files and starts the server process and any specified applications. Once the instance of Enhydra is running, the administration applications are able to perform management operations. All management operations work in the same manner; the active state (including resource parameters) of Enhydra, an application, or servlet is changed and the change may be saved in the configuration file.

When Enhydra is installed, all Enhydra applications are placed in /<ENHYDRA_ROOT>multiserver/webapps/autoload directory as war files, where <ENHYDRA_ROOT> is the directrory where Enhydra is installed. When Enhydra is started, all applications from <ENHYDRA_ROOT>/multiserver/webapps/autoload are unpacked (if needed), placed in <ENHYDRA_ROOT>/multiserver/work/webapps/<service_name>/<appName> directory and started.

NOTE: Service name value is one of Enhydra's installation options. Default value is set to Enhydra.
If an application has already been unpacked, and its war file has not been changed, the application is not again unpacked, it is only started.

JOnAS admin (url *http://localhost:<communication_port>/jonasAdmin*) is a graphical tool based on the Struts framework and the JMX technology.

NOTE: Communication port value is one of Enhydra's installation options. Default value is set to 9000.

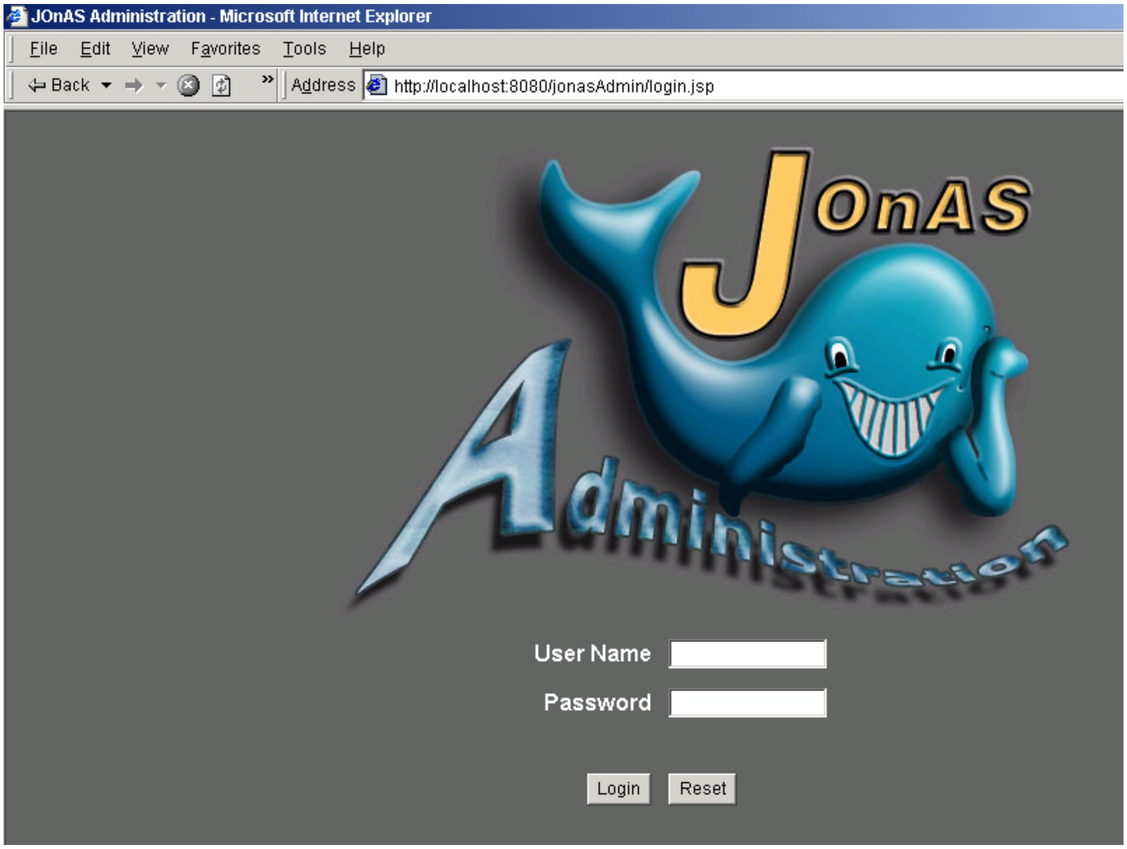

JOnAS Administration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
http://localhost:8080/jonasAdmin/login.jsp
JOnAS
Administration
User Name
Password
Login
Reset

Figure 1: JOnAS Admin

To enter JOnAS Admin, please enter User Name and Password (appropriate) values.

NOTE: Administration User Name and Password values are Enhydra's installation options. Default values are set to 'admin' (for User Name value) and 'enhydra' (for Password value).
Then you get the main JOnAS Admin window:

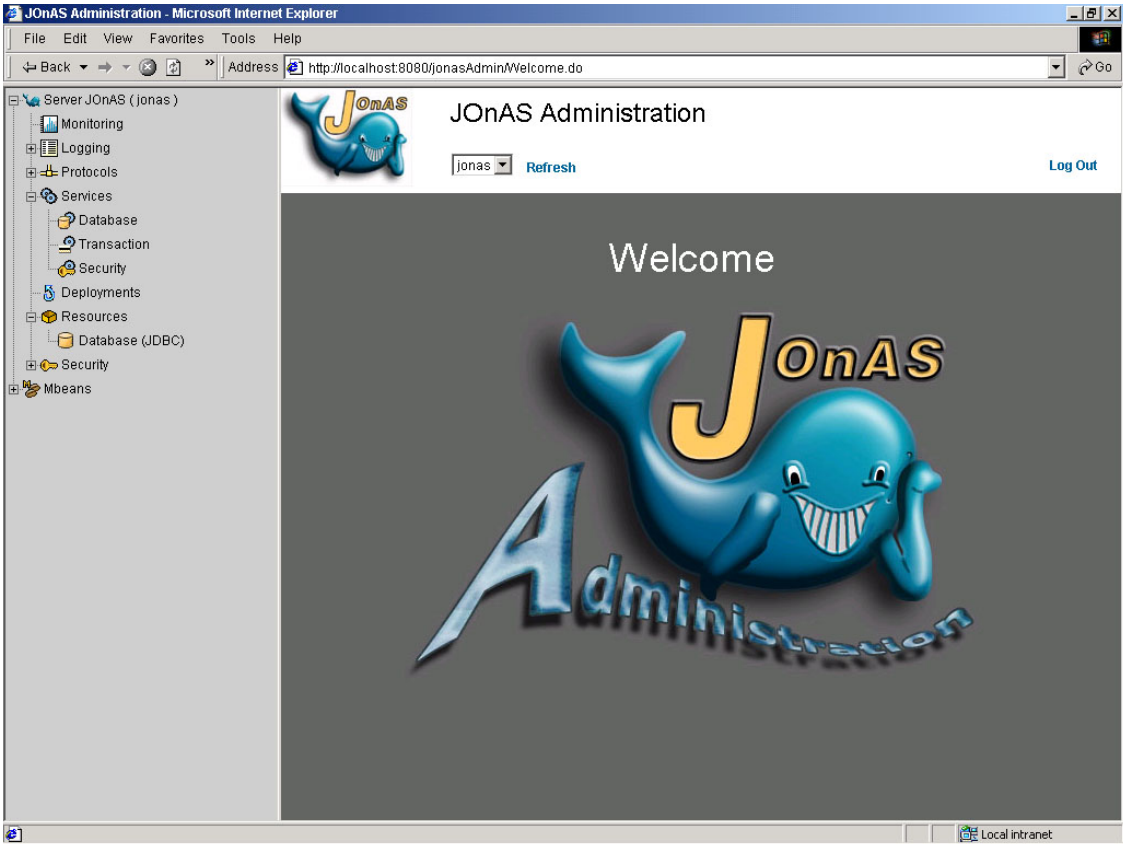


Figure 2: JOnAS Admin main window

The management tree in this window allows access to the following main management facilities:

- General information concerning the administered server
- Server monitoring
- Logging management
- Communication protocols management
- Active services presentation and configuration
- Dynamic deployment of application modules
- Resources management
- Security management

The console also allows browsing of MBeans registered in the MBean server that are associated with the currently managed JOnAS server.

Server management - displays general information about the administered JOnAS server, including the JMX server and the WEB server, and provides the capability of listing the content of the Registry.

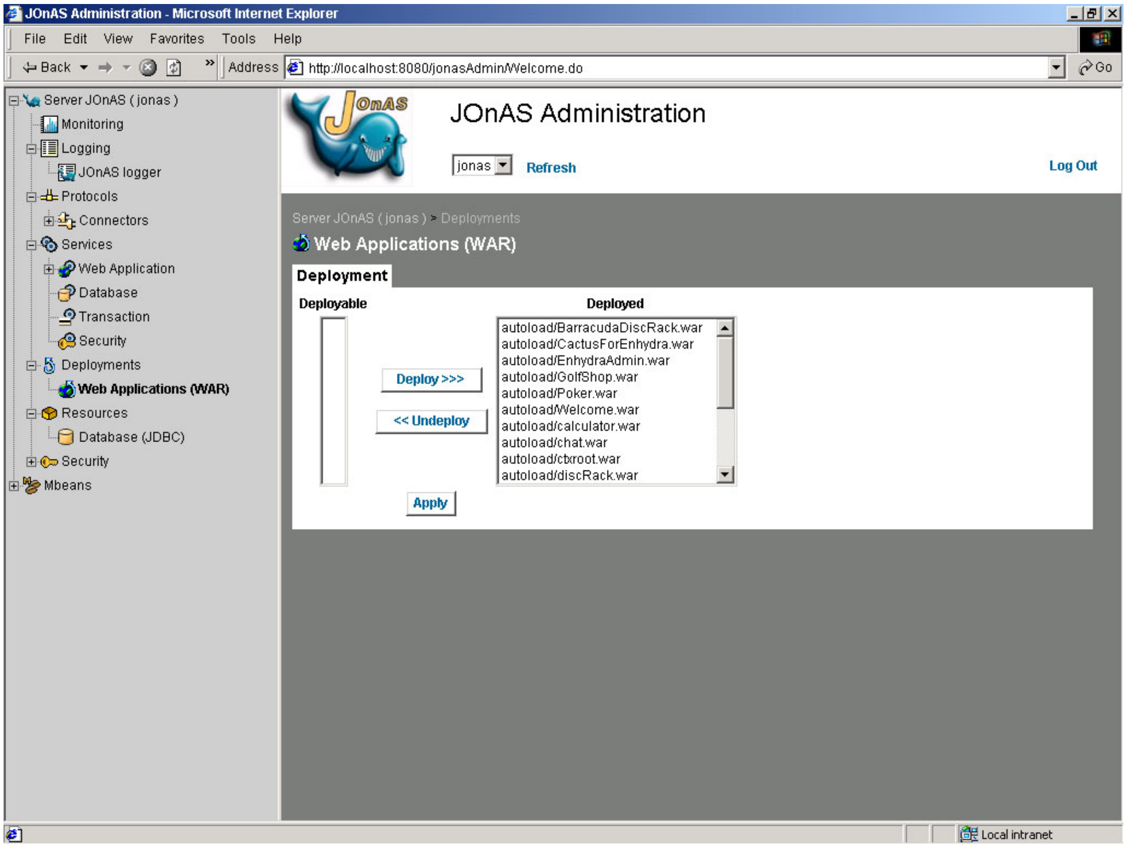
Server monitoring - presents memory usage, a count of the threads created by JOnAS, and other monitoring information concerning managed services and resources.

Logging management - allows the administrator to configure the JOnAS Logging system. Additionally, if Tomcat is used as the WEB service implementation, it allows creation of HTTP access loggers.

Communication protocols management - this facility relates to the integration of Tomcat management in JonasAdmin. It currently presents connectors defined in the Tomcat configuration and allows for the creation of new HTTP, HTTPS, or AJP connectors. Note that the Protocols sub-tree is not presented if Jetty is used as the WEB service implementation.

Active services presentation and configuration - all the active services have a corresponding sub-tree in the Services tree. Managing the various container services consists of presenting information about the components deployed in these containers. New components can be deployed using the dynamic deployment facilities presented in the next paraph. However, it may be necessary to create a new context for WEB components (WAR package) to be deployed in a Tomcat server before the deployment step, if a customized context is required by the component. This operation is performed using the New web application button. Similarly, the services that allow management of the different types of resources (DataSources, Resource Adapters, Jms and Mail resources) also provide information about the resources being deployed. Additionally, deployed resources (DataSources or MailFactories) can be reconfigured and their new configuration made persistent by using a Save button. The transaction service management allows reconfiguration (possibly persistent) and presents monitoring information about transactions managed by JOnAS.

Dynamic deployment with JonasAdmin - a very useful management operation is the capability of loading stand-alone J2EE components (JAR, WAR, RAR packages) or J2EE applications (EAR packages) using the Deployments sub-tree in the JonasAdmin console. The administrator's task is facilitated by the display of the list of deployable modules, the list of deployed modules, and the capability of transferring modules from one list to another. The deployable modules are those installed in directories specific to their type. For example, the deployable JARs are un-deployed JARs installed in JONAS_BASE/ejbjars/ or in a JONAS_BASE/ejbjars/autoload/ directory.



Address: http://localhost:8080/jonasAdmin/Welcome.do

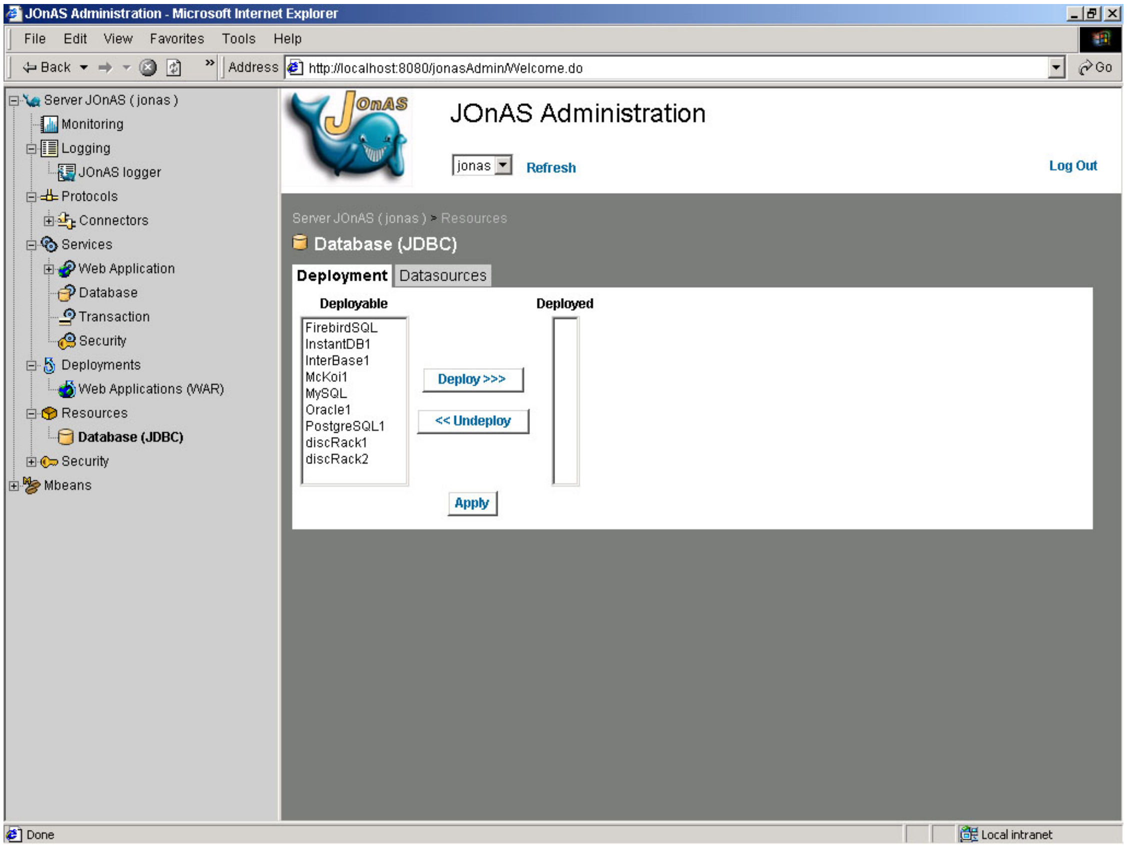
Server: jonas

Deployed:
autoload/BarracudaDiscRack.war
autoload/CactusForEnhydra.war
autoload/EnhydraAdmin.war
autoload/GolfShop.war
autoload/Poker.war
autoload/Welcome.war
autoload/calculator.war
autoload/chat.war
autoload/ctxroot.war
autoload/discRack.war

Figure 3: JOnAS Admin deployment of war files

The sub-tree *Deployments -> Web Applications (WAR)* is used for deployment and undeployment of Enhydra applications. As mentioned before in this document, Enhydra applications are placed in ENHYDRA_ROOT>/multiserver/webapps/autoload directory as war files. If you want to stop any of these applications, you have to mark that application in the list of deployed (war) applications by clicking with left mouse button, and than by clicking on *Undeploy* button, and finally, on *Confirm* button to confirm the action. The application is now successfully undeployed. Starting Enhydra application is similiar, mark that application in the list of undeployed (war) applications , click on *Deploy* button, and finally, on *Confirm* button to confirm the action. The application is now successfully deployed.

Resources management - provides the capability of loading or creating new resources managed by the active services. For example, if the JMS service is running, the JMS sub-tree in Resources presents the existing JMS destinations (Topics and Queues), and allows the removal of unused destinations and the creation of new JMS destinations. Adding or removing resources implies reconfiguration of the corresponding service. If this new configuration is saved using the Save button, the JOnAS configuration file is updated. As in the JMS service example, the removed topics are deleted from the list assigned to the *jonas.service.jms.topics* property and the newly created topics are added to this list.



FirebirdSQL
InstantDB1
InterBase1
McKoi1
MySQL
Oracle1
PostgreSQL1
discRack1
discRack2

Figure 4 JOnAS Admin database resources

The sub-tree *Resources -> Database (JDBC)* -> tab *Deployment* is used for deployment and undeployment of datasources. If you want to deploy any of these datasources, you have to mark that datasource in the list of deployed datasources by clicking with left mouse button, and than by clicking on *Undeploy* button, and finally, on *Confirm* button to confirm the action. The datasource is now successfully undeployed. Deployment of a datasource is similiar, mark that datasource in the list of undeployed datasources, click on *Deploy* button, and finally, on *Confirm* button to confirm the action. The datasource is now successfully deployed.

The sub-tree *Resources -> Database (JDBC) ->* tab *Datasources* is used for adding new datasource (by clicking on *New datasource* button), or for getting details about an existing datasource (by clicking on that datasource in the list of existing datasources).

To be able to use *Resources* sub-tree, in jonas.properties file (in <ENHYDRA_ROOT>/multiserver/conf directory), services *jtm* and *dbm* must be launched in the JOnAS Server. The propery that contains these values is *jonas.services*.

Example 12.1. Setting jonas.properties for working with database resource

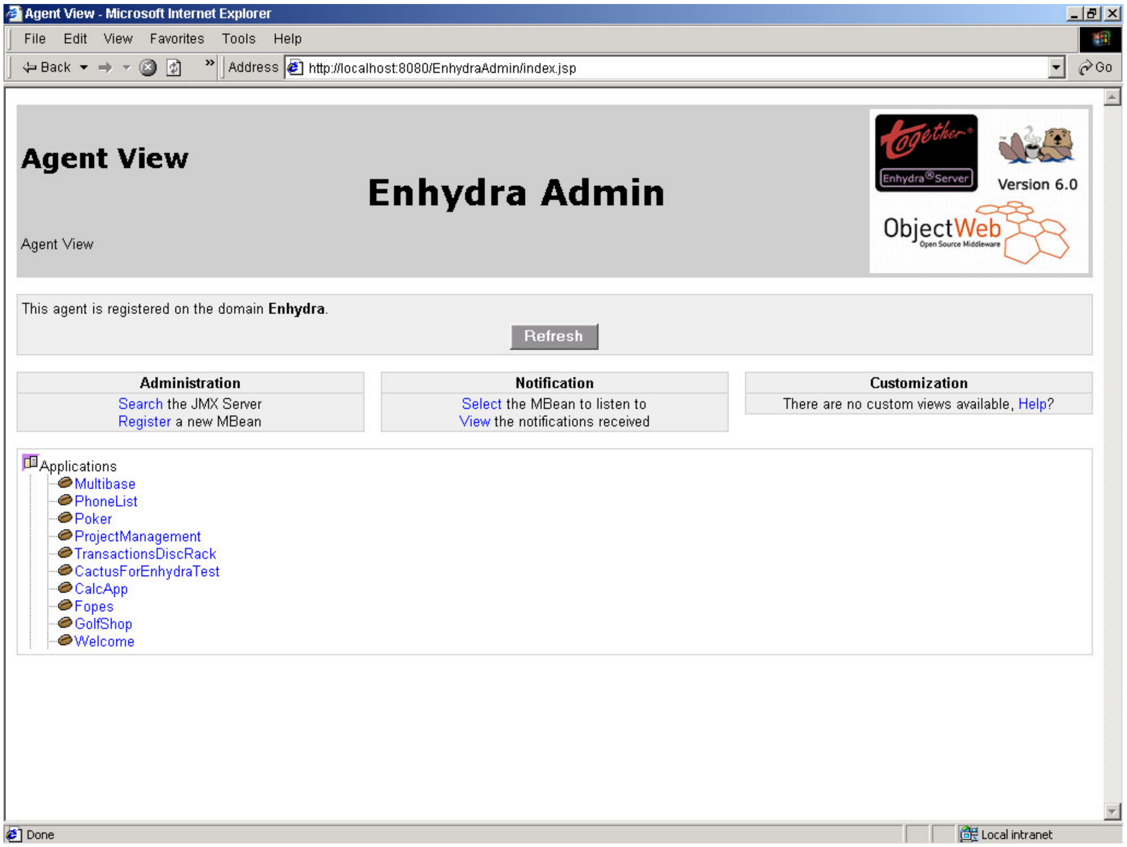
```
jonas.services jtm,dbm, security,web
```

Security management - presents existing security realms and allows the creation of new realms of different types: memory, datasource, and ldap realms.

Note concerning persistent reconfiguration facilities - it is important to note that JOnAS and Tomcat have different approaches to reconfiguration persistency. In JOnAS, every Save operation is related to a service or a resource reconfiguration. For example, the administrator can reconfigure a service and a resource, but choose to save only the new resource configuration. In Tomcat, the Save operation is global to all configuration changes that have been performed. For example, if a new HTTP connector is reconfigured and a new context created for a web application, both configuration changes are saved when using the Save button.

Enhydra Admin application (url *http://localhost:<communication_port>/EnhydraAdmin*) is a graphical tool based on the MBeans and JMX technology.

NOTE: Communication port value is one of Enhydra's installation options. Default value is set to 9000.



This agent is registered on the domain **Enhydra**.

Figure 5: Enhydra Admin main window

It is a graphical tool that enables modification of the operational attributes for Enhydra (application or servlet) and cache administration.