The OpenII Toolkit for OpenEAI

Release 3 – July 2007 – DRAFT

Table of Contents

(items in the table of contents are hyperlinks)

What this guide covers 3
About the OpenII Toolkit for OpenEAI 3
What's new 6
Installation and setup 7
The Administrative Console 7
The Login Page 8
Authentication 8
JAAS Configuration 9
Authorization 10
User Maintenance 11
Site-specific Branding 12
Specifying Configuration Defaults 14
Importing the Core Services 16
Package Descriptors 19
The Consumer Details Page 32
The Scheduled Application Details Page 38
The Schedule Details Page 41
The Command Details Page 43
The Routing Service 61
The Routing Service Page 62
The Router Targets Page 66
The Add Router Target Page 68
The Select Routing Criteria Page 70
The Proxy Service Page 72
Add Proxied Application Page 77
Proxied Application Maintenance Page 78
Targets this Application can Send Requests to 80
Add New Proxy Target Page 81
Select Target Proxy Criteria Page 81
The Logging Service Page 84
ELS Consumers Tab 87
What happens when an application or gateway is started? 88
Reports/Administrative views 89

The Transformation Service 89
The RDBMS Connector Suite 89
The File Connector Suite 89
The Test Suite Application 90
The Message Object Generation Application 90
The Console Configuration Parameters 91

What this guide covers

This book discusses the OpenII Toolkit for OpenEAI. It describes the components included in the toolkit and how to install, use, and administer those components.

About the OpenII Toolkit for OpenEAI

The OpenII Toolkit for OpenEAI (a.k.a., the toolkit) is a suite of integration infrastructure components and applications, which is…

- Comprised of enhanced and/or re-implemented versions of the OpenEAI Project router, request proxy, and logger, with user interfaces that enhance ease of use and ease of administration.
- A set of general connectors for popular classes of technologies that must participate in an integrated enterprise.
- Packaged and documented so it can be deployed in a basic configuration with minimal effort for easy demonstration and initial use by untrained staff.
- Packaged and documented so that expanding the deployment for higher-volume, clustered deployments is easy for trained staff.
- Tested and certified to run on relevant platforms.
- Tested and certified to run with relevant message transports.
- Accompanied by useful example integrations.
- Supported with an annual schedule of enhancements and platform and infrastructure recertification to keep pace with new infrastructure requirements and operating system and message transport upgrades.

The OpenII Toolkit for OpenEAI includes enhanced or re-implemented versions of all of the OpenEAI Project reference implementations (see www.openeai.org). It also includes three new general purpose connectors for commonly-used technology.

OpenII Administration Console for OpenEAI

This is an application and agent infrastructure that allows organizations using OpenEAI to view the status of, start, and stop all OpenEAI applications from one web-based user interface. It uses the existing Java Management Extensions (JMX) features of OpenEAI, but employs a new agent and application for presenting these administration features to users.

This console also provides a simple mechanism to manage, generate, and publish important OpenEAI artifacts which must be available to OpenEAI applications at runtime, such as XML primed documents, Message Object APIs, and Enterprise Object documents, and OpenEAI deployment descriptors. This allows organizations presently running separate applications to generate and separate web servers to publish these artifacts to manage them centrally.

Once all of these artifacts are managed and generated by a central application using a specific hierarchy it is then possible to generate prolific amounts of accurate, detailed documentation about the integrations within an enterprise. For example, documentation such as that prepared by the University of Illinois to document its messaging enterprise at http://www.aits.uillinois.edu/

GetEnterpriseData can be generated, along with descriptions of which applications are authoritative for which message objects. The initial release of the OpenII Administration Console for OpenEAI sets the stage for this documentation generation by providing the central repository, or registry, for all of these artifacts, and generates some of this basic documentation. Subsequent releases of the OpenII Administration Console for OpenEAI will focus on exploiting all possibilities for automatic enterprise documentation generation.

OpenII Enterprise Documentation Application for OpenEAI

This application generates the enterprise integration documentation outlined above, in a web-publishable format. It is most frequently used from within the administration console to generate documentation for artifacts deployed from or registered with the administration console.

OpenII Request Proxy Service for OpenEAI

This application includes the functionality of the existing OpenEAI request proxy reference implementation and adds the ability to implement the more complex proxy functions of filtering fields and pluggable, complex proxy rules. The field filtering feature allows applications which may only be authorized to see certain fields of an enterprise object to use the pre-existing message support of an authoritative source by stripping restricted fields out of messages sent by the authoritative source before forwarding them to restricted requesting applications. The complex proxy rules provide a pattern for performing any arbitrary logic or subsequent messaging as part of making the decision on whether or not to proxy a specific message. This request proxy will also be improved to be fully manageable from within the console application. Subsequent releases of the request proxy will focus on providing a way to enter user-defined proxy rules from within the administration console.

OpenII Synchronization Message Routing Service for OpenEAI

This application includes the functionality of the existing OpenEAI synchronization message router reference implementation and adds the ability to be fully managed from the administration console.

OpenII Logging Service for OpenEAI

This application includes the functionality of the existing OpenEAI logging service reference implementation and adds the ability to be fully managed from the administration console. The administration console provides web interfaces for viewing all logged synchronization messages and republishing a selection of messages to any endpoint in the enterprise. It also provides a web interface for viewing synchronization error messages and correlating them with the synchronization messages that precipitated the error. The administration console functionality for the logging service provides an interface for developing basic reports about logged messages and logged error messages. Subsequent releases will focus on additional messaging reporting capabilities and a synchronization error alert mechanism.

OpenII Test Gateway for OpenEAI

This application includes the functionality of the existing OpenEAI Any ERP Vendor Gateway example implementation and adds the ability to be fully managed from the administration console. It can quickly become a “straw man” authoritative source for any enterprise object.

OpenII Test Suite Application for OpenEAI

This application includes the functionality of the existing OpenEAI test suite application reference implementation and adds the ability to be fully managed from the administration console. The

administration console features for the test suite application include managing multiple configurations and test suites, so specific configuration and test suite documents can be quickly selected and used to execute tests from the administration console.

OpenII Poller Application for OpenEAI

This application includes the functionality of the existing OpenEAI destination poller reference implementation and adds the ability to poll synchronization-message-consuming gateways by publishing synchronization verification messages. The administration console features for the poller application provide a mechanism for reporting on whether or not the synchronization verification messages are being properly verified by synchronization-consuming gateways.

OpenII Message Object API Generation Application for OpenEAI

This application includes the functionality of the existing OpenEAI MOA generation application and adds the ability to be run from within the administration console using message definitions that are registered with the administration console.

OpenII File Connector for OpenEAI

The File Connector for OpenEAI is a suite of analysis artifacts, runtime artifacts, and applications that provide general foundation to perform the following tasks:

- Provide mappings, translations, and transformations among enterprise objects and extracts.
- Read records from extracts, build enterprise objects from the content of the extracts, and publish sync messages based on that content. Generally, these files will be built and dropped-off by legacy systems: the general goal is that the transactions represented in the extracts be applied at some other interested source.
- Consume synchronization messages from an authoritative source and build an extract file from the content of those messages (i.e., using the enterprise objects contained within those messages). This extract can then be processed by a legacy system that already expects to process these types of files.

These components are general infrastructure. This means they will be used as often as possible to process many different extracts and to use many different enterprise objects. The infrastructure is not built with any one specific extract in mind.

OpenII Database Connector for OpenEAI
(Not available in release 1, will be added in a later release)

The Database Connector for OpenEAI is a suite of analysis artifacts, runtime artifacts, and applications that provide general foundation to perform the following tasks:

- Provide mappings, translations, and transformations among enterprise objects and the tables of a relational database.
- Detect changes in the state of objects mapped to database data, read data from the relational database, build enterprise objects from the data in the database, and publish the appropriate synchronization messages.
- Consume synchronization messages and apply them to the relational database.
- Provide request/reply message support for objects for which the relational database system is authoritative.

These components are general infrastructure. This means they will be used as often as possible to expose many different relational database systems. The infrastructure is not built with any one relational database system in mind.

OpenII Transformation Service for OpenEAI

The Transformation Service for OpenEAI is responsible for consuming messages published or produced by one system, 'system A', and transforming them into messages that are specific to another system, 'system B'. The transformed message is then sent on to interested targets. In the request-reply modality (synchronous) the transformation service will also transform the response sent by the target system into the appropriate response for the requesting application.

OpenII Web Service Connector for OpenEAI
(Will be added in a later release)

The Web Service Connector for OpenEAI is a suite of analysis artifacts, runtime artifacts, and applications that provide mappings, translations, and transformations among messages and the invocations of web services. This connector can be used to enable request/reply messaging with applications exposed via web services or to consume synchronization messages and apply them to applications that are exposed via web services.

These components are general infrastructure. This means they will be used as often as possible to expose many different applications that expose their logic and data using web services. The infrastructure is not built with any one application or service in mind.

What's new

Release 3 in April 2007 is the second release of the toolkit. The following components are included in release 3:

- The OpenII Administration Console for OpenEAI (***updated***)
- The OpenII Synchronization Message Router for OpenEAI (***updated***)
- The OpenII Request Message Proxy for OpenEAI (***updated***)
- The OpenII Logging Service for OpenEAI (***updated***)
- The OpenII File Connector suite for OpenEAI (***updated***)
- The Documentation Application for OpenEAI (***updated***)
- The OpenII Database Connector for OpenEAI (***updated***)
- The OpenII Test Gateway (***updated***)
- The OpenII Transformation Service (***new***)
- The OpenII Test Suite application (***new***)
- The OpenII Message Object Generation application (***new***)

There are other components included that are being released via the LGPL software license, as allowed by the OpenEAI Software Foundation. They include:

- The Poller Application

The LGPL applications can be administered and run via the console but they are not fully integrated into the console at this time. Commercial versions of these applications will be included and fully integrated into the console in future releases.

Finally, the following components do not yet exist but will be available and integrated into the console in future releases:

- The OpenII Web Service Connector for OpenEAI

Installation and setup

See the Getting Started Guide for detailed installation and setup steps.

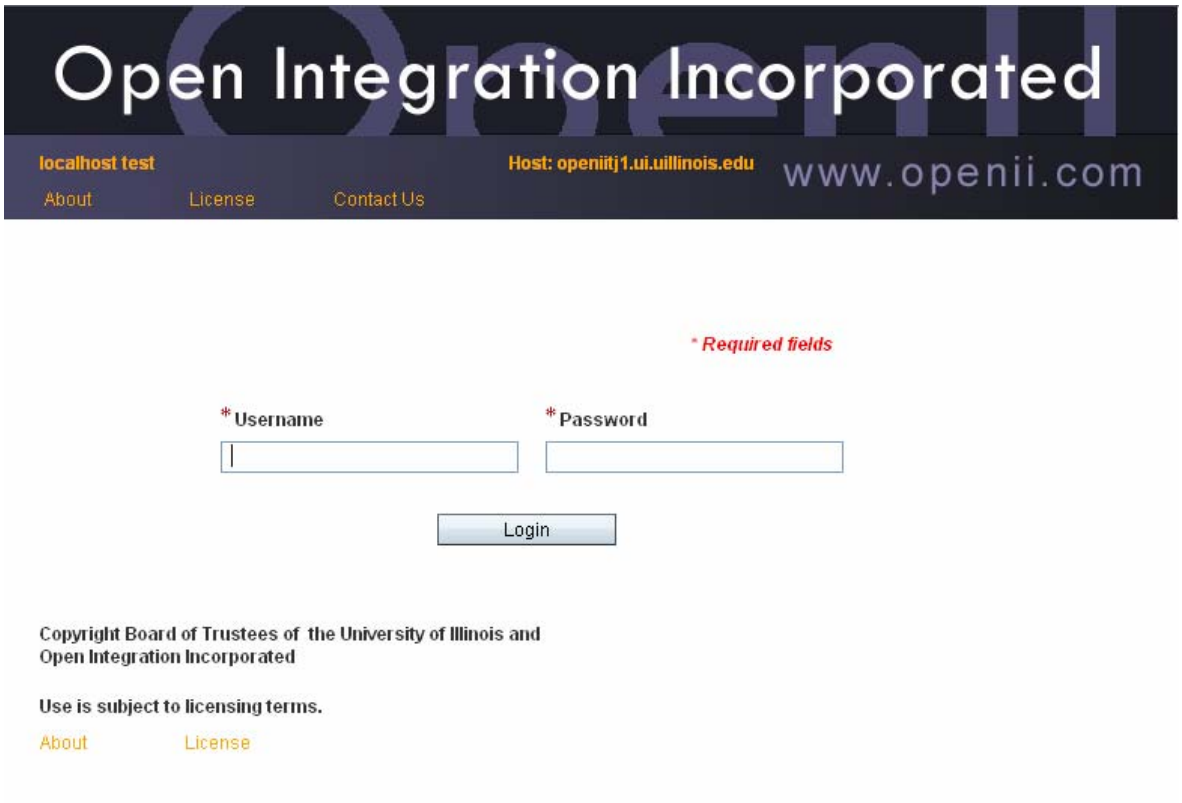
The Administrative Console

The OpenII Administrative Console is a web-based application that provides users the ability to configure and run OpenEAI-based applications and gateways via a point-and-click user interface. It is specifically designed to ease and improve the administration and maintenance of OpenEAI-based applications. In addition to providing an interface to manage any OpenEAI-based application or gateway, the Console also provides specific interfaces for certain OpenII infrastructure applications, such as the Routing Service, Proxy Service, Logging Service, Transformation Service, Documentation Application, File Connector Suite, and the RDBMS Connector Suite.

Once the toolkit is downloaded, installed and all supporting components are running (i.e., the database and application server) you can log into the application to start managing and running the applications that are distributed. Currently, there are several sample applications distributed with the Toolkit that can be managed via the Console. As new integrations are identified and the analysis is performed to implement those integrations, new applications and/or gateways will be added via the console. Then, those applications and gateways can be managed and run via the console in the same manner as the samples that are provided.

To open up the console, open a browser and go to the following URL: http://servername:port/consoleq4. Note, this URL may vary based on the application server being used to run the console and deployment choices made during the deployment. The Toolkit distribution ships with Tomcat and by default the URL is: `http://servername:8181/consoleq4`.

The Login Page



The first page that appears is the `Login Page`. By default, the Toolkit ships with three console users (`user1`, `user2` and `user3`). These users are all members of the directory server that ships with the Toolkit distribution (OpenLDAP). By default, `user1` and `user2` are known Console Administrators. `User3` is an existing directory server user but has not been registered as a console user so by default, you must add this user before you can log in as them. This process is walked through in the getting started guide. All users share the same default password of "***password!***".

These users are simply provided for demonstration purposes and to confirm that the typical installation was successful. Administration of an organization's directory server is not something covered by this document but the console does have the ability to authenticate against any LDAP or Active Directory based directory.

Enhanced

Authentication

There are really two steps to authentication of a user. The first step is binding to a directory to authenticate the user against credentials associated to that user in the directory. The console does use Java Authentication and Authorization Service (JAAS) however, in version 3.0, the only JAAS implementation available for authentication is LDAP/AD. The 4.0 release slated for Q32008 will include additional implementations for other external authentication infrastructure.

With release 3.0, users may be authenticated against multiple trees in a single directory. So, if some administrators or users are in the `ou=Examples,ou=Users,dc=any-openeai-enterprise,dc=org` tree of the organization's directory and others are in the `ou=Examples,ou=People,dc=any-openeai-enterprise,dc=org` tree, the console will consult both trees during initial authentication. The search base that the console should use for authentication is specified in two places:

1. In the `jaas.conf` file that is deployed in the application server
2. In the `web.xml` file that is associated to the console instance being run

JAAS Configuration

`jaas.conf` – should contain the following entry which describes the authentication method and implementation as well as the configuration required for that implementaiton. The values will be site-specific but basically, these parameters are telling the JAAS implementation being used what directory to connect to (URL), the search base(s) to attempt to bind to (SEARCHBASE) and the bind filter that should be appended to the user name supplied by the user when they attempt to log in (BIND_FILTER).

```
ldapAuth {
      com.openii.toolkit.jaas.LdapLoginModule required
      URL= "ldap://localhost:389"
      SEARCHBASE= "ou=Examples,ou=Users,dc=foo,dc=com"
      BIND_FILTER="uid=";
};
```

If multiple trees of a directory should be used for initial authentication, the SEARCHBASE attribute listed above would contain a semi-colon delimited list of valid trees in the directory. Example:

```
SEARCHBASE=
"ou=Examples,ou=Users,dc=foo,dc=com;ou=Examples,ou=People,dc=foo,dc=com"
```

`web.xml` – should contain the following `context-param` entries. Their values will be site specific but must match the corresponding values listed above in the `jaas.conf` file. These are the ONLY `context-param` entries required in the `web.xml` and in future releases they will not be included in `web.xml` at all. When the consoleq4.war file is exploded the first time, this file will be laid down as is common for any web application of this type. Once the file exists on the file system, administrators can modify it and the `jaas.conf` file referred to provide the site-specific values required for the particular environment the console is running in.

```
<context-param>
  <description>URL to the LDAP server.  NOTE:  This must also be
specified in the jaas.conf file (or other appropriate JAAS
configuration)</description>
  <param-name>ldapHost</param-name>
  <param-value>ldap://localhost:389</param-value>
</context-param>

<context-param>
  <description>semi-colon delimited set of distinguished names that map
to the location in a directory server where user authentication will
```

```
occur.  NOTE:  This property must also be specified in the jaas.conf file
(or other appropriate JAAS configuration).</description>
  <param-name>searchBaseDn</param-name>
  <param-value>ou=Examples,ou=Users,dc=foo,dc=com</param-value>
</context-param>

<context-param>
  <description>The directory attribute that will be appended to the user
name that the user logs in as.  NOTE:  this must also be specified in the
jaas.conf file (or other appropriate JAAS configuration).</description>
  <param-name>bindFilter</param-name>
  <param-value>uid=</param-value>
</context-param>
```

See the Console Configuration Parameters section of this document for more information about these and other configuration parameters and their purpose.

If the user types an incorrect User Name or Password, an error will be displayed on the screen that indicates there was an error and provides instructions to the user depending on the nature of the error.

Once the user is successfully authenticated against the directory, the console consults its database to see if the user is a known console user and what role(s) are associated to the user for authorization. If the user is successfully authenticated against the directory but they **are not** a known "console user" an error message will be displayed indicating this (see figure x). Console users can be maintained from within the console (see the Authorization section below).

Figure x – the user 'deptuser1' was successfully authenticated against the directory but they are not a registered user of the Console.

Authorization

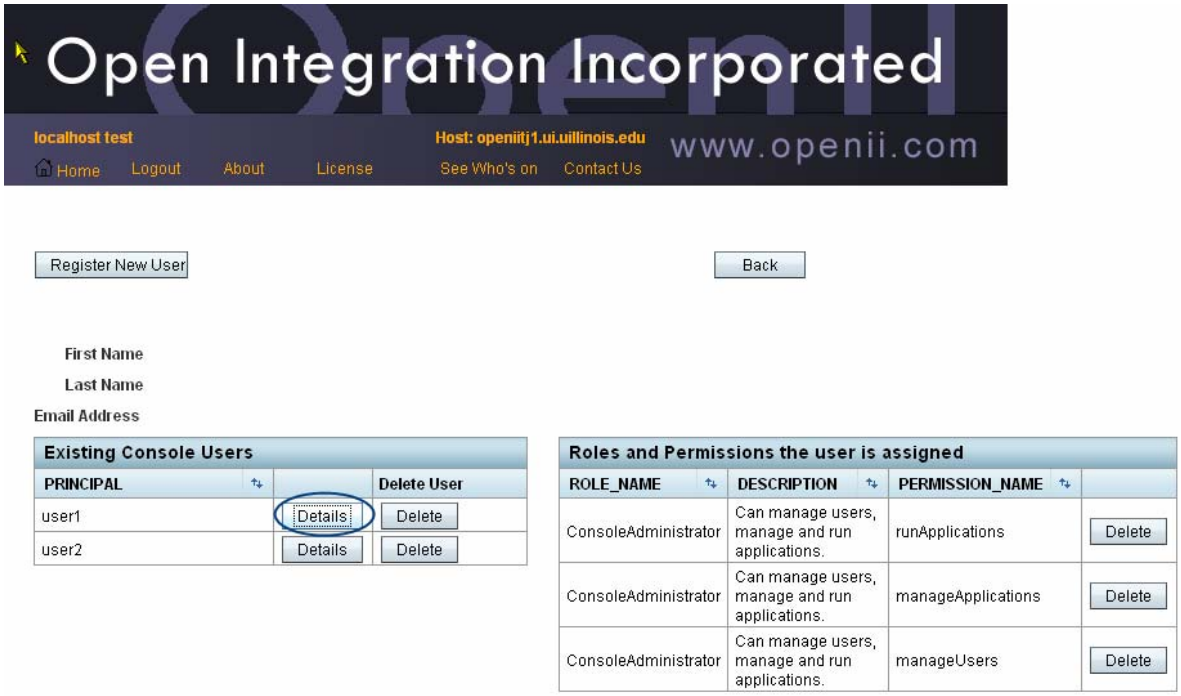
For release 3.0, authorization is controlled internally by the console. In this release, there are two levels of authorization available:

- ConsoleAdministrator – which is allowed to manage and run applications as well as manage other users of the console.
- ApplicationAdministrator – which is allowed to manage and run applications but cannot manage other users.

In version 3.0, the only difference between these two users is that ConsoleAdministrators can register and maintain users via the User Maintenance link on the General Tab which will allow them to log into the console and mange applications. In order to register a new user, that user must exist in the directory that the console is configured to authenticate against (see Authentication). ApplicationAdminstrators can simply administer applications.

User Maintenance

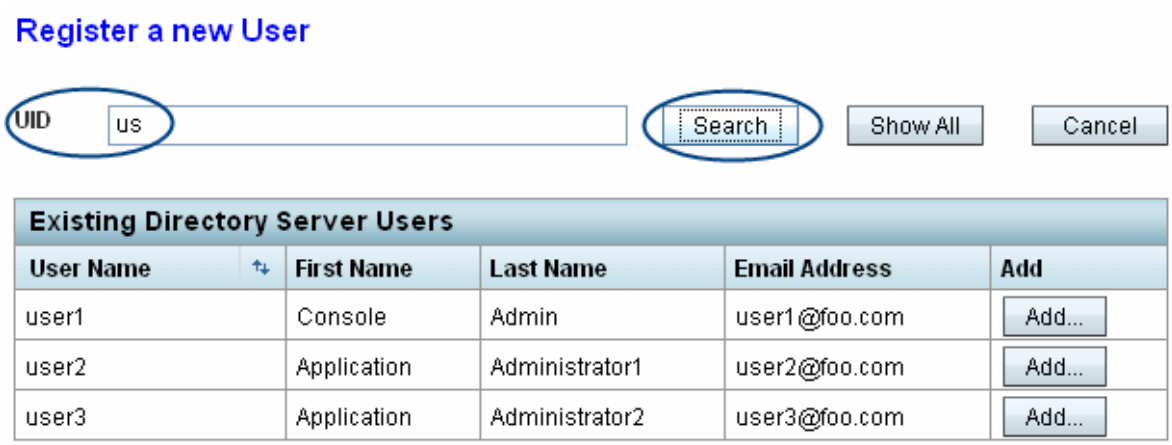
When a ConsoleAdministrator follows the User Maintenance link on the General tab, they are taken to the page seen below. Here, they may view users currently allowed to use the console in some capacity. By clicking on the Details button, they can view and/or modify roles and permissions are assigned to that user.



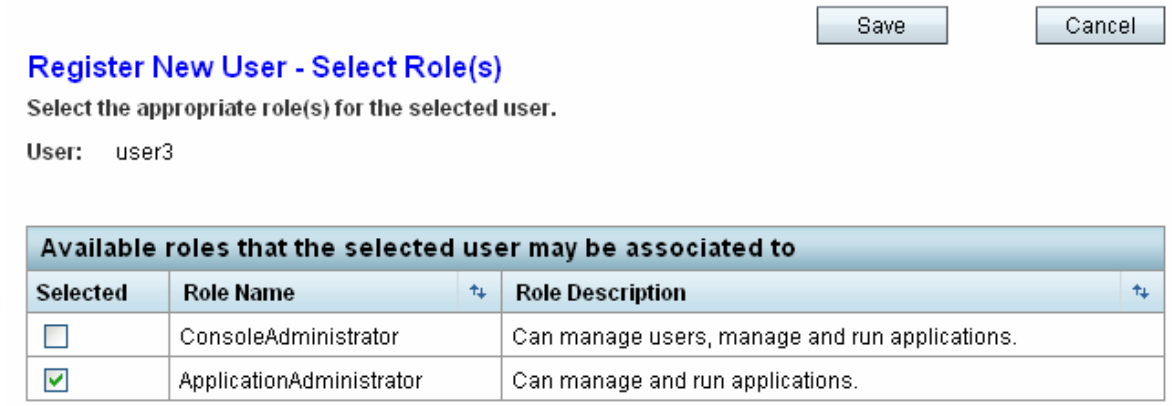
Registering a new user

To register a new user, a ConsoleAdministrator clicks on the “Register New User” button on the User Maintenance page. This will take the administrator to a page where they may search a directory for potential users that might be added. The directory tree that is searched is based on the settings specified above in the Authentication section (URL/ldapHost, SEARCHBASE/searchBaseDn and BIND_FILTER/bindFilter).

The administrator may enter a full or partial UID (directory User Name) into the UID search field and click the Search button to narrow the list of items presented.



Once the user that should be added it found, the administrator may click on the Add… button to select the Role that should be associated to that user. As mentioned before, in release 3.0, the only roles available are “ConsoleAdmininstrator” and “ApplicationAdministrator”. Once the role is selected, clicking the Save button will associate that user to the selected role and the user will be allowed to log into and use the console. At this time, the only difference between the two roles is that ConsoleAdministrators may access this portion of the application (User Maintenance) and ApplicationAdministrators may not.



[TODO – discuss how to configure the console to use someone other than user1, user2 or user3 the first time]

Site-specific Branding

Vendors and/or site administrators may specify the header image that should be used by the console. The header image is an organization-specific image that can be used to give the application a specific “look and feel” for the organization that is using the console or for vendors that may be including the console with another suite of products. The configuration parameter that controls which header image to use is called `headerImagePath`. The value associated to this parameter should be the fully qualified path to the image that you wish to use. For more

information about this and other configurable parameters and their purpose, refer to the Console Configuration Parameters section of this document.

Figure x - Example header image which can be replaced with an image specific for your site.

Specifying Configuration Defaults

During installation of the console, several runtime configuration properties are established that are used by applications maintained via the console as well as by the console itself. Many of these parameters are "general" and apply to any environment regardless of choices made by an organization that uses the console and the other components of the Toolkit. However, all configuration parameters that may be site-specific must be specified the first time a new instance is brought online. The console knows if it has been fully configured previously and if it has not, it will prompt the user to provide information about these site-specific configuration parameters.

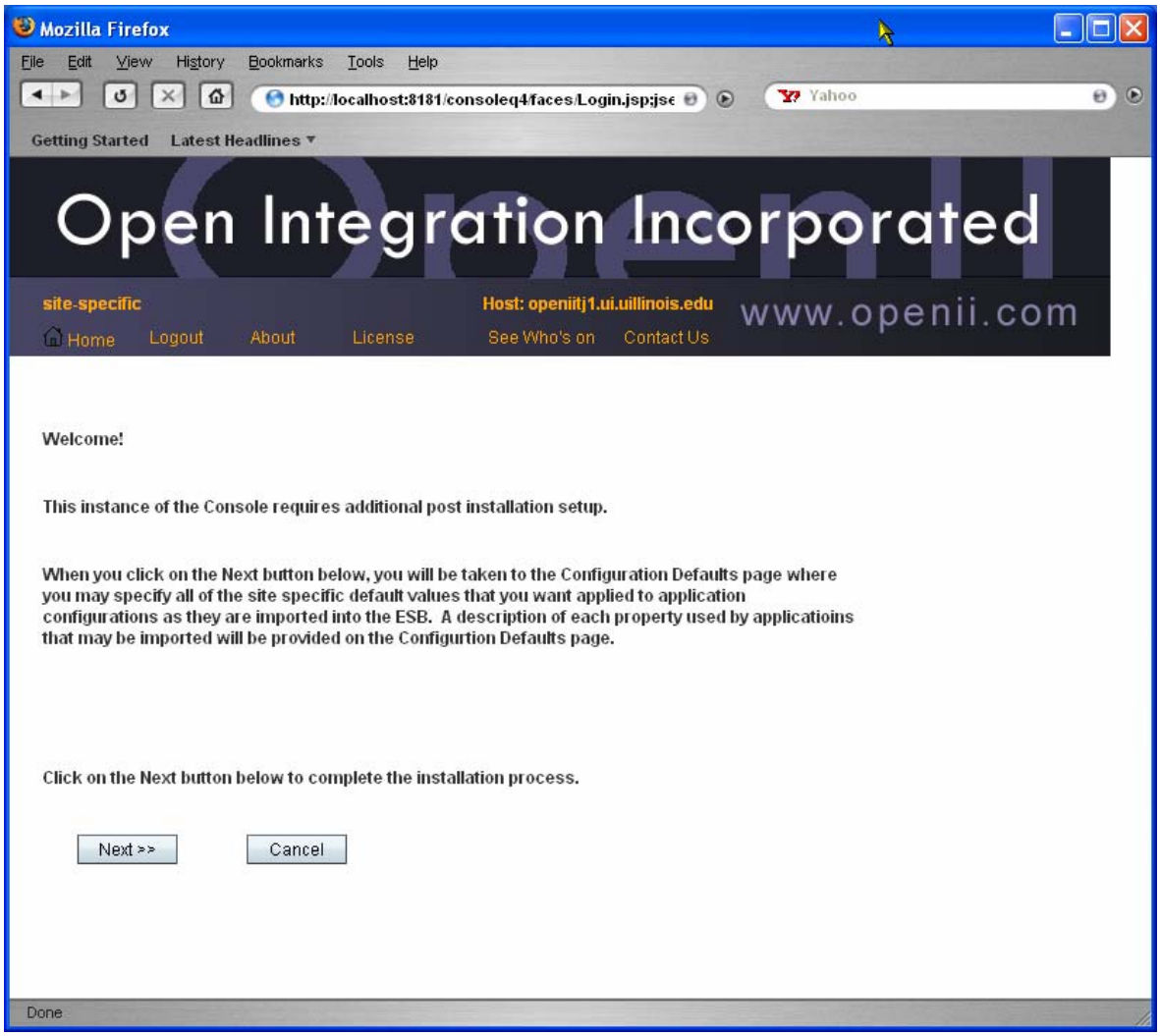


Figure x – the console recognizes that this instance has not been fully initialized so it will prompt the user to provide post-installation information regarding the site-specific configuration parameters.

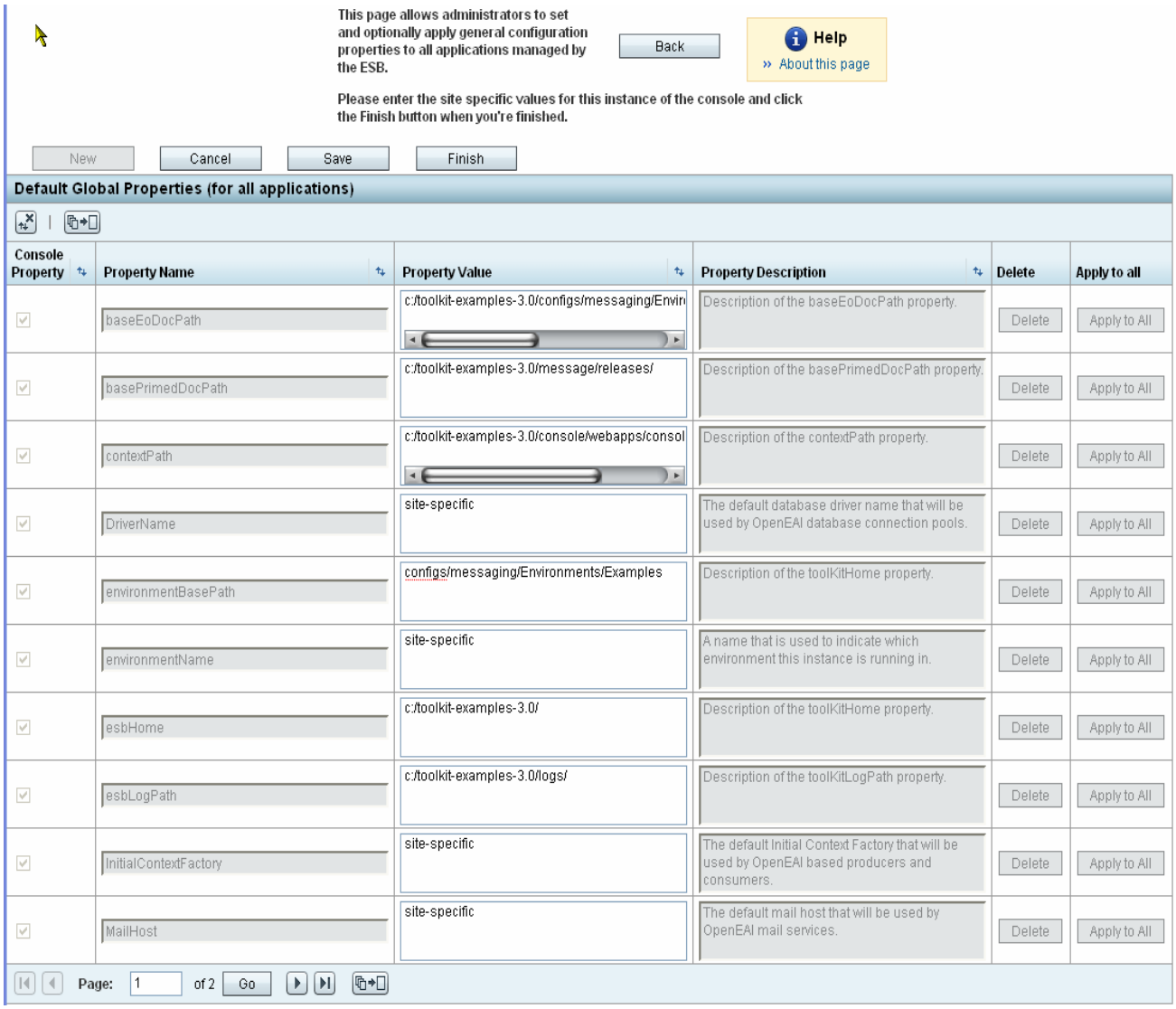


Figure x – Administrators specify information that is unique to their environment that the console can use during application configuration and management to make the user experience more efficient.

This interface is used during initial "post-installation" setup but can also be accessed via the Configuration Defaults link from the General tab. At any time, administrators can come to this page and change values associated to these parameters. Once a value is changed, the new value can be propagated to all applications that use the relevant property. For example, if during initial post-installation setup, a value of `com.foo.jdbc.Driver` was used for the `DriverName` property but at some point in the future it was decided to use a different "default" JDBC driver implementation, the administrator could come to this page and change the `DriverName` value from `com.foo.jdbc.Driver` to `com.foo.jdbc.v2.Driver` and click the Apply to All button to have that change propagated to all applications that use database connections.

During post-installation setup, the following site-specific properties must be specified (see the Console Configuration Parameters section of this document for more information about these parameters):

- `DriverName`
- `environmentName`
- `InitialContextFactory`
- `MailHost`
- `ProviderUrl`

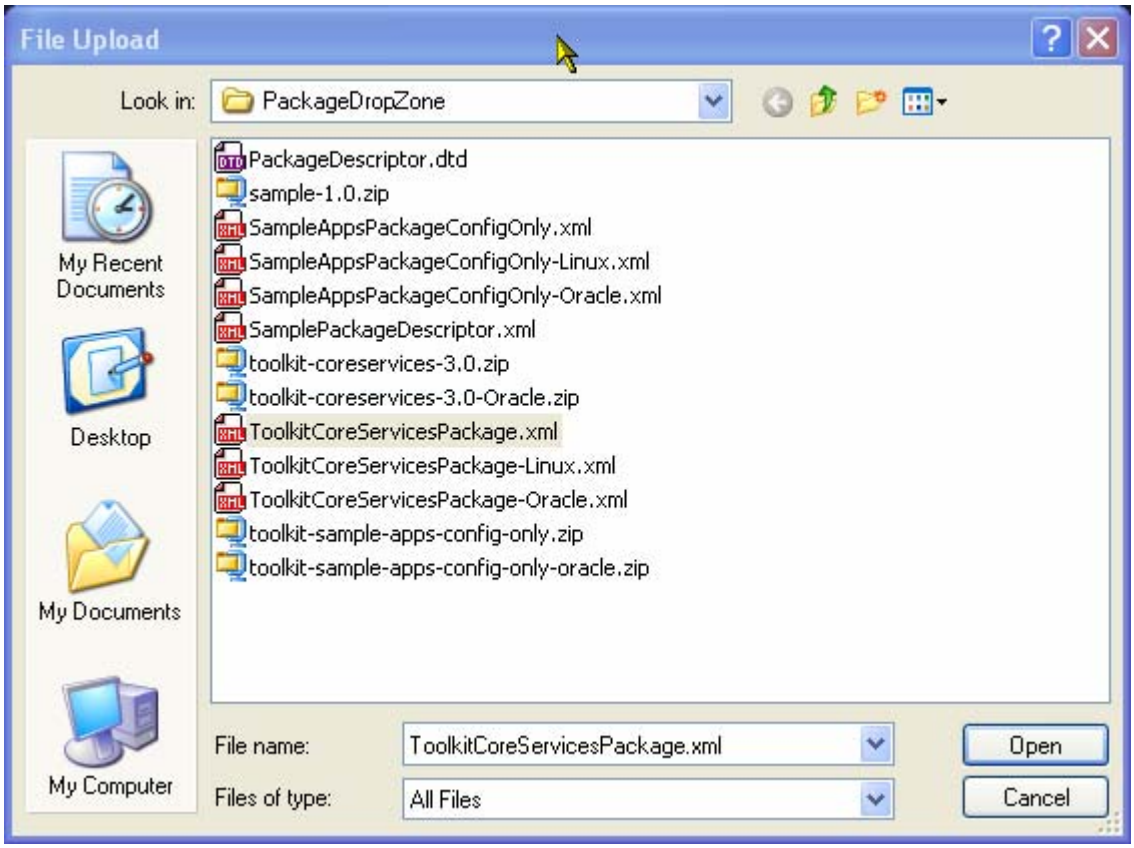
Once all site-specific properties have been specified and saved, the user will click on the Finish button to continue. During post-installation setup, clicking on the Finish button will take the user to the Import Applications page, because this is an "empty" ESB at this point (i.e., none of the core services exist yet). On the Import Applications page, the user will be given the opportunity to import the core services and any other applications that they wish to import at this time.

Importing the Core Services

The figure below shows the Import Applications page. The user is taken to this page the first time an instance of the console is brought online to "install" the Core Services as well as any other applications that may be of interest to the organization from the "Sample Enterprise" that is usually included with a distribution of the ESB. Additionally, administrators can access this page by following the Import link on the General Page when/if additional applications need to be deployed into the ESB.

The general process for importing an application (or set of applications) is as follows:

1. The user clicks the Browse button and locates a “package descriptor” which describes the content of the package to be imported. A package descriptor is an XML artifact that describes all of the components related to the application(s) being imported. This may include: configuration artifacts, libraries, sample messages, enterprise objects documents which are all part of every OpenEAI based application. See the Package Descriptor section below for more information regarding the purpose and structure of a package descriptor and how they are used by the Console.



2. Once the user has located the appropriate package descriptor, they click on the Upload File button. The console then reads the package descriptor and lists all applications that are described in that package descriptor.
3. At this point, the user can optionally choose to search and replace anything that might need to be changed in the source artifacts described by the descriptor but this should be performed with caution. Generally, an application administrator will know the content of the descriptor and know whether or not any replacements need to occur.
4. The user may optionally modify the “ESB Home” that should be associated to the package being imported. Generally, a package is created in one environment and this process is deploying it into another environment. Because of this, it may be necessary to tell the console the source “ESB Home” so it knows what value to replace with the target ESB Home which is a default parameter established when performing the post-installation steps related to the default configuration parameters above.
5. The user may also specify information about the “configuration base” associated to the source package so that value may be replaced with the value currently associated to this instance of the ESB.
6. If desired, the user can click the Select All button to select all applications contained in the package or they can select individual applications from the package to import if they do not wish to import all applications contained in the package.
7. Finally, by clicking the Import Selected Applications button, the ESB will process the package descriptor and import all artifacts associated to each application included in the package. While it’s doing this, it will automatically replace certain key configuration parameters with values that have been specified during the post-installation steps related to the default configuration parameters mentioned above.

[todo: More general information here about package descriptors and their purpose etc.]
Purpose of a package descriptor
Content
Variations (config only, config plus other artifacts, pointing directly to a deployment descriptor/config doc)

Package Descriptors

Package descriptors are XML artifacts that describe the content of an application or suite of applications that will be imported into the runtime environment of an instance of the ESB. They are similar to .war files or .ear files which describe the runtime components needed for an application server to run a standard J2EE web application for example. These package descriptors provide information about the runtime artifacts required to run an OpenEAI based application and the ESB "imports" configuration and other runtime artifacts from the package to the appropriate places in the runtime environment.

Generally, when a new connector or service is built, or when an instance of a general connector, like the RDBMS Connector needs to be deployed, it is tested in a non-production environment before it is deployed to a production environment. Once the connector or service is deemed "ready" for production, it should be packaged up to make deployment into that production environment most efficient. Once the package is ready, it can be imported using the same steps as described above in the **Importing Core Services** section of this document. It is recommended that a structured approach like this be followed to ensure consistency during deployment.

There are several different types of package descriptors that may be used and examples of each are included with the standard Toolkit sample enterprise that licensed users can download, install and use as a starting point. The various types of packages and artifacts that can be imported are described below. The type of package created by a deployment administrator depends on the application being deployed and the state of the target runtime environment it will be deployed into.

Standalone configuration document
The simplest way to import an application into the ESB is by importing that application's configuration from an existing OpenEAI deployment descriptor (a.k.a., config doc). During the import process, administrators can simply point to an existing deployment descriptor and select it. The console will determine which applications exist in that descriptor and present the user with a list of all applications. Administrators can then select the application(s) they wish to import and click the "Import Selected Applications" as described **above**. When this happens, the configuration for this application will be imported and stored in the console's database. This is useful when all other required artifacts already exist in the target runtime environment and nothing else needs to be copied over. A typical use of this type of import is when an organization is upgrading from a previous version of the ESB or when an application that has never been run from within the console needs to be brought in.

Named Application(s), configuration only from a package

XML
DOCTYPE PackageDescriptor
PackageDescriptor
= sourceEsbHome c:/toolkit-examples-3.0/
Archive
= name c:/PackageDropZone/toolkit-coreservices-3.0.zip
Description The Enterprise Service Core (ESC) distribution package.
Application (3)

	= id	= type	= configDoc
1	com.openii.SyncRouter	Router	Console-Router.xml
2	com.openii.LoggingService	LoggingService	Console-ELS.xml
3	com.openii.RequestProxy	Proxy	Console-Proxy.xml

Named application(s), configuration and other artifacts from a package

XML
DOCTYPE PackageDescriptor
PackageDescriptor
= sourceEsbHome c:/oldesbhome-3.0/
Archive
= name c:/PackageDropZone/sample-1.0.zip
Description A sample Connector distribution package.
Application
= id com.foo.IdCorrelationService
= type General
= configDoc IdCorrelation.xml
Directory (1)

	Source	Target
1	docs/api	c:/toolkit-examples-3.0/Documentation/javadoc/

File (3)

	Source	Target
1	idcorrelation.jar	c:/toolkit-examples-3.0/Jars/4.0/
2	ObjectIdCorrelationEO.xml	c:/toolkit-examples-3.0/EnterpriseObjects/com/foo/Core/1.0/
3	ObjectIdCorrelationQuerySpecificationEO.xml	c:/toolkit-examples-3.0/EnterpriseObjects/com/foo/Resources/1.0/

All applications, configuration and other artifacts from a package

XML
DOCTYPE PackageDescriptor
PackageDescriptor
= sourceEsbHome c:/oldesbhome-3.0/
Archive
= name c:/PackageDropZone/sample-1.0.zip
Description A sample Connector distribution package.
Content
= configDoc Console.xml
= type General
Directory (3)

	Source	Target
1	docs/api	c:/toolkit-examples-3.0/Documentation/javadoc/
2	jars	c:/toolkit-examples-3.0/Jars/4.0/
3	suites	c:/toolkit-examples-3.0/TestSuites/

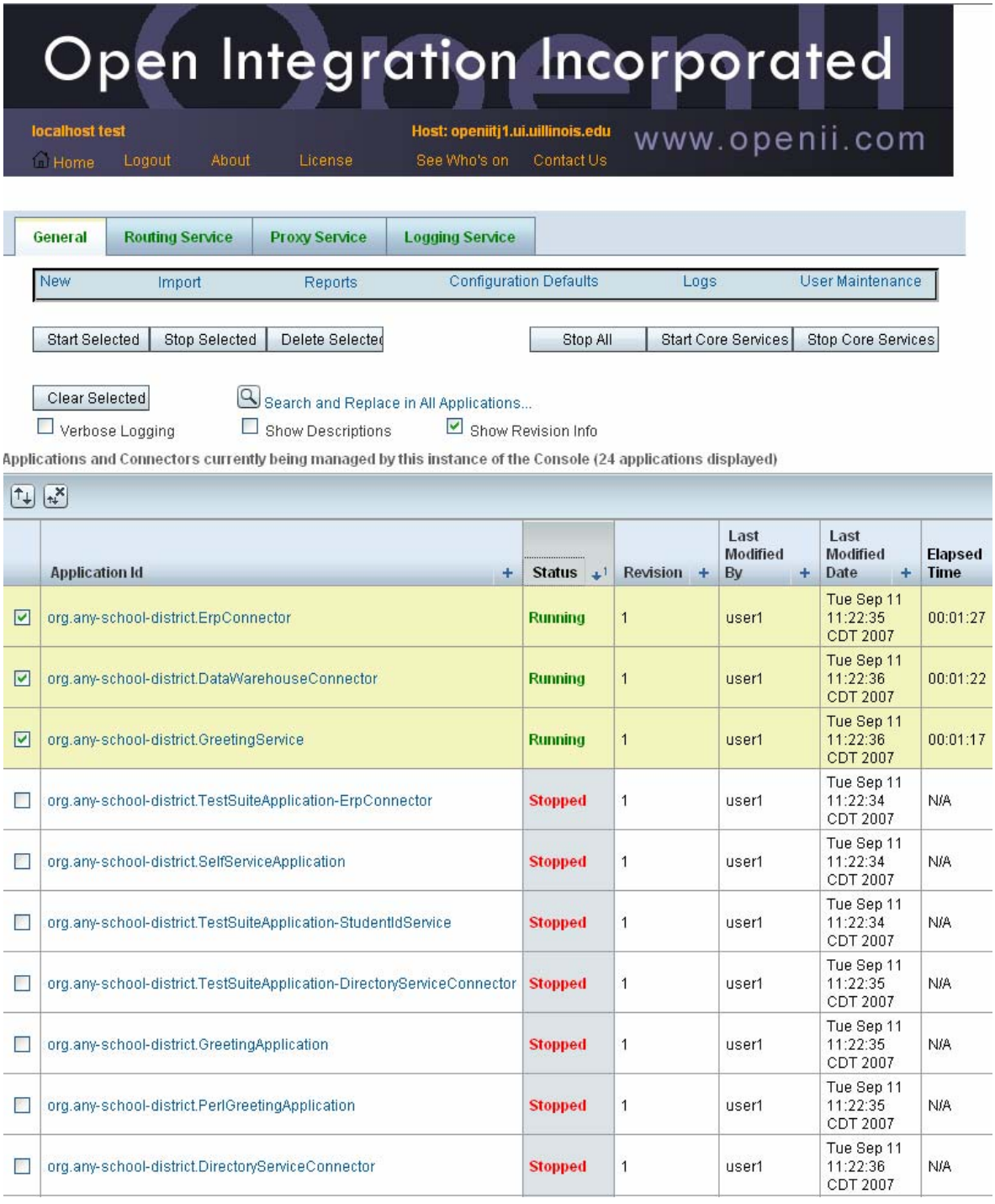
File (4)

	Source	Target
1	AddressEO.xml	c:/toolkit-examples-3.0/EnterpriseObjects/com/foo/Resources/1.0
2	NameEO.xml	c:/toolkit-examples-3.0/EnterpriseObjects/com/foo/Resources/1.0/
3	PersonEO.xml	c:/toolkit-examples-3.0/EnterpriseObjects/com/foo/Person/1.0/
4	create_mysql.sql	c:/toolkit-examples-3.0/Databases/transformsvc

Enhanced

The General Tab

Once a user logs in successfully and performs the initial post-installation setup, this is the first page they see.



Application Id	Status	Revision	Last Modified By	Last Modified Date	Elapsed Time
org.any-school-district.ErpConnector	Running	1	user1	Tue Sep 11 11:22:35 CDT 2007	00:01:27
org.any-school-district.DataWarehouseConnector	Running	1	user1	Tue Sep 11 11:22:36 CDT 2007	00:01:22
org.any-school-district.GreetingService	Running	1	user1	Tue Sep 11 11:22:36 CDT 2007	00:01:17
org.any-school-district.TestSuiteApplication-ErpConnector	Stopped	1	user1	Tue Sep 11 11:22:34 CDT 2007	N/A
org.any-school-district.SelfServiceApplication	Stopped	1	user1	Tue Sep 11 11:22:34 CDT 2007	N/A
org.any-school-district.TestSuiteApplication-StudentIdService	Stopped	1	user1	Tue Sep 11 11:22:34 CDT 2007	N/A
org.any-school-district.TestSuiteApplication-DirectoryServiceConnector	Stopped	1	user1	Tue Sep 11 11:22:35 CDT 2007	N/A
org.any-school-district.GreetingApplication	Stopped	1	user1	Tue Sep 11 11:22:35 CDT 2007	N/A
org.any-school-district.PerlGreetingApplication	Stopped	1	user1	Tue Sep 11 11:22:35 CDT 2007	N/A
org.any-school-district.DirectoryServiceConnector	Stopped	1	user1	Tue Sep 11 11:22:36 CDT 2007	N/A

This tab presents the user with a navigation tab set. On the `General Page`, the `General` tab will be selected. The user may reach the `General Page` by either:

1) Successfully logging in, or
2) Clicking on the "General" tab from the tab set
3) Clicking on the Home link

The `General Page` lists all OpenEAI-based components that are not part of the Enterprise Service Core (ESC), i.e. not the Routing Service, Proxy Service, Logging Service, Documentation Application, etc. These core services, part of the OpenII Tool Kit for OpenEAI, have dedicated interfaces in the Console.

The `General Page` provides the following high-level functions:

- **Retrieves information about the applications and gateways already being managed by the console** (those running and those available to run) based on existing configuration documents that have been used to either start or view applications.
- **Builds a list of applications from that configuration information** that the user can manage.
- **Displays that list of components and each component's status**, i.e., if it is running, stopped, etc. Also provides the description of the application/gateway which can be displayed by checking the `Show Description` check box.
- **Builds a link based on the application/gateway ID** that when clicked on will take the user to a new page where the user can manage the application (see `Application Details` page below). When a component is selected, the `Application Details` page will be opened and the user will be able to modify configuration information related to the application. Here, the user can manage consumers associated to the application, manage the scheduled applications associated to the application and view/modify log information that the application writes to.
- **Displays the amount of time the application has been running (if it's running)**

One of the core functions of the console is to configure and manage OpenEAI based applications, gateways, connectors, services, etc. The configuration actions performed via the console ultimately result in the creation or modification of a standard OpenEAI deployment descriptor (a.k.a., the config doc) that is used by all OpenEAI based applications to configure themselves. These documents contain all the information required by an application in order to perform the business logic associated to them. In-depth details regarding OpenEAI deployment descriptors are available in the OpenEAI API Introduction document on the OpenEAI Project website.

With release 3.0 all application configurations are stored in a database and the console is configured via standard J2EE based configuration practices to point to that database. It reads the database to determine which applications it is currently managing and when changes are made to the configuration of an application, those changes are stored back to the database. When an application, or group of applications, are imported into the console via the Import Application process described above, the relevant components from the package associated to those applications is stored in the console's `T_APP_CONFIGS` table. This table is created during installation and stores each application's configuration as a single row. Each time a change is made to the configuration of an application and it's saved, a new row is written to the

T_APP_CONFIGS table for that application. This provides the ability for the console to display a "revision history" for each application it is managing. Future versions of the console will provide even more "version control" type features to allow an administrator to view the differences between certain revisions of an application's configuration, restore a previous version etc. In version 3.0 the administrator simply has the ability to see who made changes to an application's configuration at a given time.

Below is an overview of some of the new or enhanced features on this page

- **Links**
  - New
    - Takes the user to the Specify General Application Information
  - Import
    - Takes the user to the Import Application page
  - Reports
    - Takes the user to the Documentation Service page
  - Configuration Defaults
    - Takes the user to the Configuration Defaults page
  - Logs
    - Takes the user to the Log Maintenance page
  - User Maintenance
    - Takes the user to the User Maintenance page
- **Buttons**
  - Start Selected
    - Starts the selected application(s) in the application list
  - Stop Selected
    - Stops the selected application(s) in the application list
  - Delete Selected
    - Deletes the selected application(s) in the application list
  - Stop All
    - Stops all applications that are running, including the Core Services
  - Start Core Services
    - Starts the Core Services: `com.openii.SyncRouter`, `com.openii.RequestProxy`, and `com.openii.LoggingService` all together.
  - Stop Core Services
    - Stops the Core Services: `com.openii.SyncRouter`, `com.openii.RequestProxy`, and `com.openii.LoggingService` all together.
  - Clear Selected
    - Clears selected applications from the application list
- **Search and Replace**
  - Use this interface to perform mass "search and replaces" in the configurations of applications being managed by this ESB. IMPORTANT: This action should be performed with extreme caution
- **Verbose Logging**
  - Select to generate more detail in the application log
- **Show Descriptions**
  - Select to display a description of the application in the application list
- **Show Revision Info**
  - Select to display a Revision, Last Modified Date and Last Modified By of the application in the application list
- **Table Columns**

- Application Id
  - The application name/Id that has been assigned to the application at setup.
  - It is recommended that these IDs be as descriptive as possible so anyone who sees the ID can make some assumptions as to the nature of the application.
- Status
  - Run status of the application (Running, Stopped, Error)
- Revision
  - How many times the application has been modified
- Last Modified by
  - Who last modified the application
- Last Modified Date
  - The date the last modification was saved
- Elapsed Time
  - Time that the application took to execute

The Application Details Page

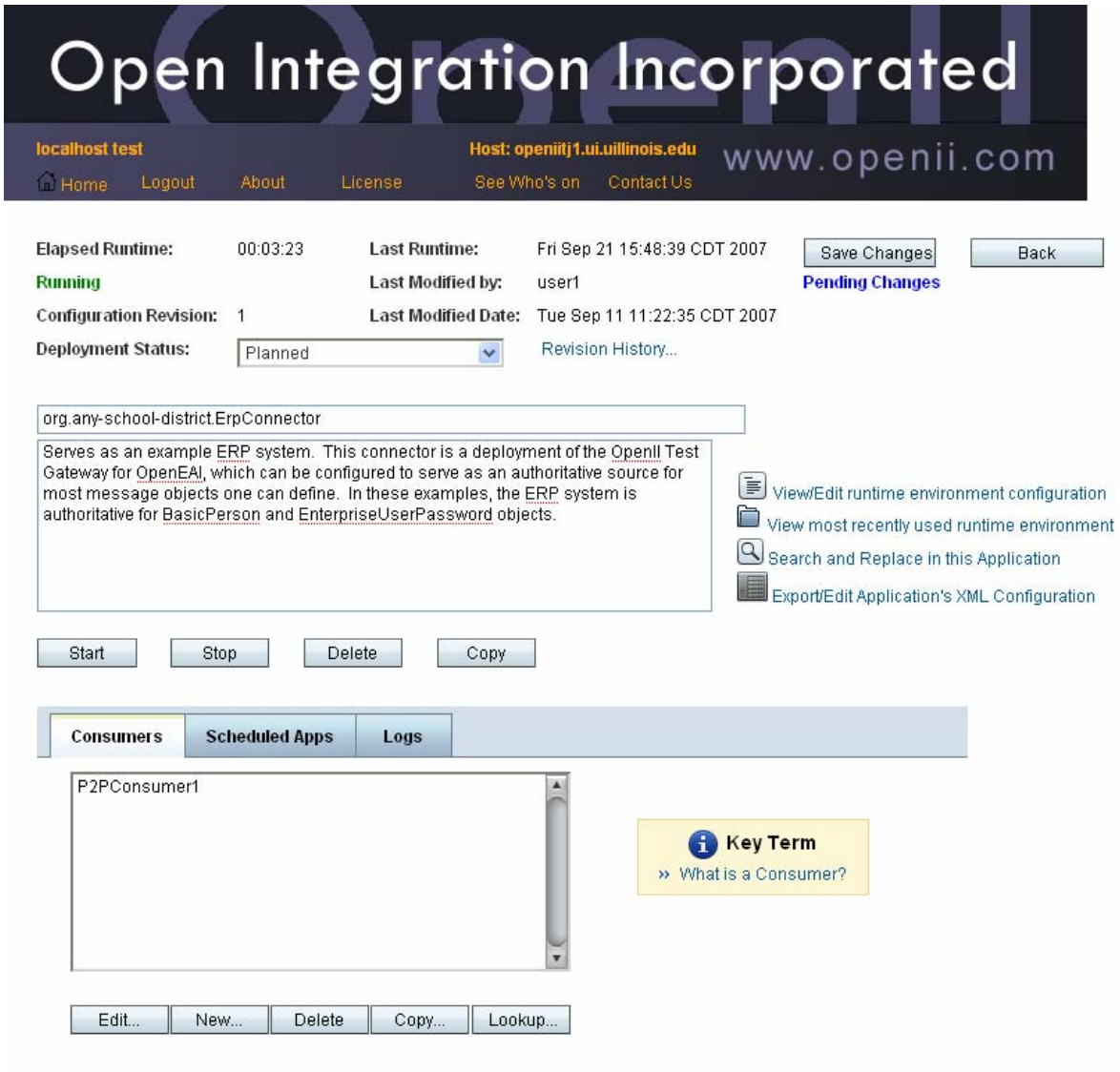
When the user clicks on a specific application link on the `General Page`, they are taken to the `Application Details Page`. The `Application Details Page` is the first in a series of pages that allow the user to manage OpenEAI based applications/gateways.

The console treats every application listed in the General Config Document as an application. In OpenEAI terms, applications that consume messages and execute business logic based on the message consumed are called Gateways. Applications that execute business logic based on a schedule are called Scheduled Applications. This screen presents the user with a list of both message consuming and scheduled applications.

Gateways are Java applications that consume messages from JMS topics or queues and perform some business logic depending on the message consumed. A gateway is made up of several pieces. There is a process (i.e., the JVM – Java Virtual Machine), a consumer that connects to a topic or queue, and one or more commands that are executed when a message is consumed by the consumer. The command is where the application logic code actually resides. It's a Java class that implements a certain set of business logic, based on integration analysis. So, from the console's perspective, when maintaining a gateway, you're telling the gateway which consumers to use, what topics and/or queues those consumers to connect to, and what business logic (via Commands) to execute when a message is consumed. For in-depth information about gateways, refer to the OpenEAI API Introduction Document.

Scheduled Applications are Java applications that perform some business logic based on a configurable schedule. When the schedule is met, the business logic is executed via similar commands as mentioned above in the Gateway discussion. The only difference between a Scheduled Application and a Gateway is the method in which the business logic execution is triggered. Business logic associated to gateways is executed when a message arrives on a JMS Topic or Queue. The gateway's consumer consumes the message, determines which command to execute and invokes the business logic in that command. Business logic associated to scheduled applications is executed when a schedule is met. The schedule is met, the scheduled app determines which command to execute and invokes the business logic in that command. That schedule can be configured via the console and tells the application when to execute the business logic. Schedules can be of several types:

- simple schedules like "execute the business logic immediately, when the application starts"
- a bit more complex like "executed the business logic every n seconds" where n is the amount of time in between the execution of the business logic and can be specified via the console
- even more complex like "execute the business logic every Monday, Wednesday and Friday at 1:00PM". This particular type of schedule CANNOT be configured via the console in this release but will be available in future releases.



On the `Application Details Page`, the user is presented with several pieces of information regarding the selected application, including:

- **The "ID" associated with the application**, which the user may change. Application IDs are important pieces of information that can be used to better organize your enterprise. So, some thought should be given as to the types of IDs that should be used. Generally, it is recommended that these IDs be as descriptive as possible so anyone who sees the ID can make some assumptions as to the nature of the application.
- **The description associated with the application**, which the user may also change. Like IDs, descriptions are important. They provide a convenient way to describe the high-level function of the application and can be very useful in documenting your enterprise. If clear and informative application IDs and descriptions are used, applications such as the Documentation Application can use this information to present a clear picture of your organization that can be shared among departments and with partners. This type of

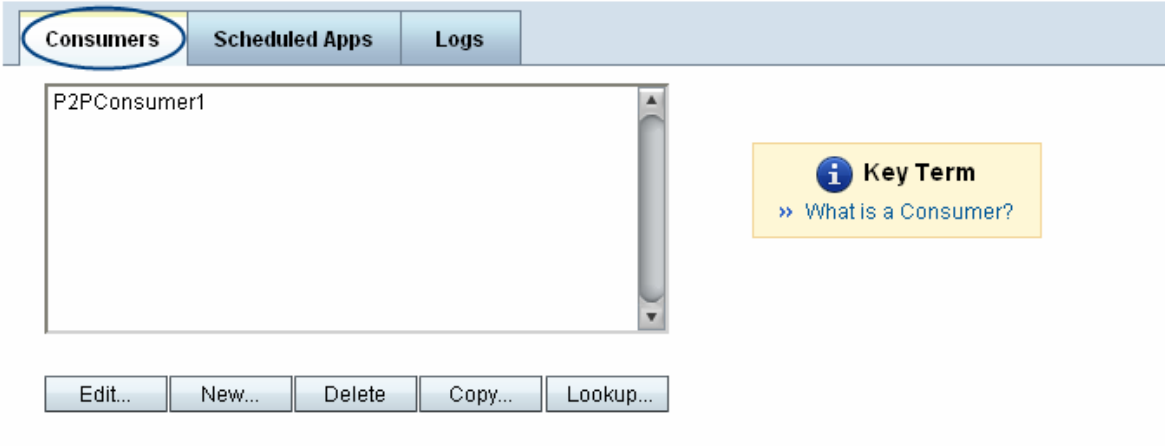
"picture" can be very valuable when performing things such as integration analysis, impact analysis, and simply communicating about what an organization does.

- **The status of the application** (**Running** or **Stopped**).
- **A list of consumers associated to the application** which, when selected, may also be modified on subsequent pages (further details on this are presented below). As mentioned above, a gateway is a Java application that consists of one or more OpenEAI foundation objects called "Consumers." OpenEAI consumers connect to JMS topics or queues and execute specific business logic when a message is consumed off of that topic or queue.
- **A list of scheduled applications associated to the application** which, when selected, may also be modified on subsequent pages (further details on this are provided below). As mentioned above, the scheduled application foundation is Java foundation that allows certain business logic to be executed on a particular schedule.
- **View and configure the log** associated to this application.

In addition to this information that the user may manage, other "application-specific" functions are available. The user has the ability to perform the following functions related to the selected application on the `Application Details Page`:

- **Start the application by clicking on the "Start" button.** When you start a gateway or application managed by the console, it will initiate a new native process to run that application in. Because of this, it is important to remember that ***when you start a gateway/application from within the console, you should always stop that gateway/application using the console***. This will prevent the creation of troublesome "orphan" processes that will remain running even when/if the console is closed or even if the application server is stopped. This behavior may vary based on the operating system being used, but in general, any gateway/application that is started via the console should also be stopped via the console. Refer to the what happens when an application or gateway is started section of this document for more information.
- **Stop the application by clicking on the "Stop" button.** This will terminate the process associated to the running gateway that is currently selected.
- **Create a copy of the current application** and create a new application that looks exactly like this gateway (named differently) then they can make changes to that gateway to make it unique.
- **Delete the current application** (as long as it is not running) by clicking on the "Delete" button.
- **Rename the current application** by clicking on the "Rename" button.
- **Save any changes made to the configuration of the application** via any of the other screens associated to managing a application (discussed later). Then, when the application is restarted, it will use the new settings.

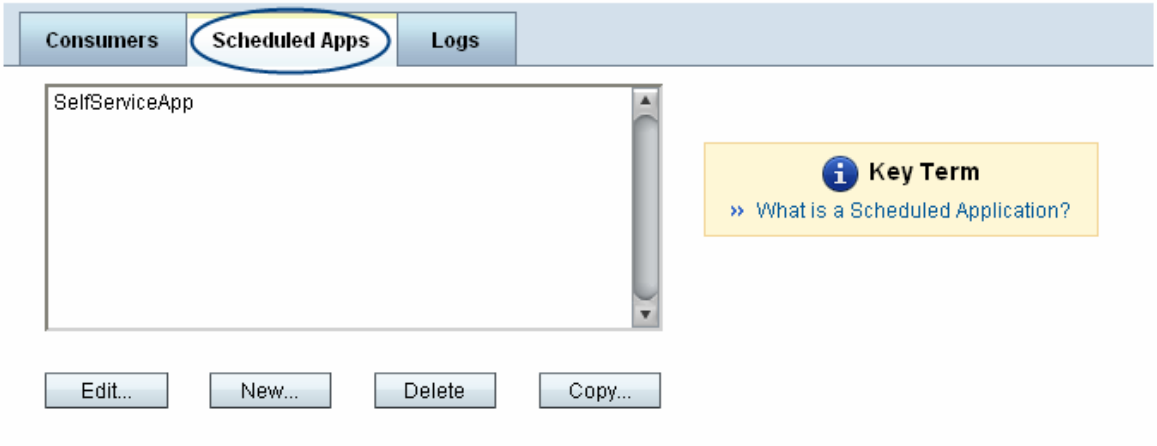
As mentioned above, if present, a list of consumers associated to this application is presented on this page.



Once the user has selected a consumer from the list, they may perform the following actions on that consumer:

- **Edit the consumer's configuration** by clicking on the "Edit…" button. When the user clicks on the "Edit…" button, they will be taken to the `Consumer Details Page` (see Consumer Details section below) where they can change specific information related to the consumer itself. The user may also double click on a consumer from the list to be taken to the `Consumer Details Page`.
- **Create a new consumer from scratch** by clicking on the "New…" button which will walk the user through a "wizard" for creating a new consumer.
- **Delete the selected consumer** by clicking on the "Delete…" button.
- **Create a copy of an existing consumer** that looks exactly like the selected consumer by selecting the "Copy…" button. When a consumer is copied, the user will be taken to the `Consumer Details Page` where they can change the consumer's name (required) and other consumer configuration information as they see fit (such as the queue or topic to which it is connecting, what commands are to be executed, etc.).
- **Lookup consumers** by clicking on the "Lookup" button. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Component page.` The user can then select from the dropdown list of consumers used by other applications to either Copy or Edit.

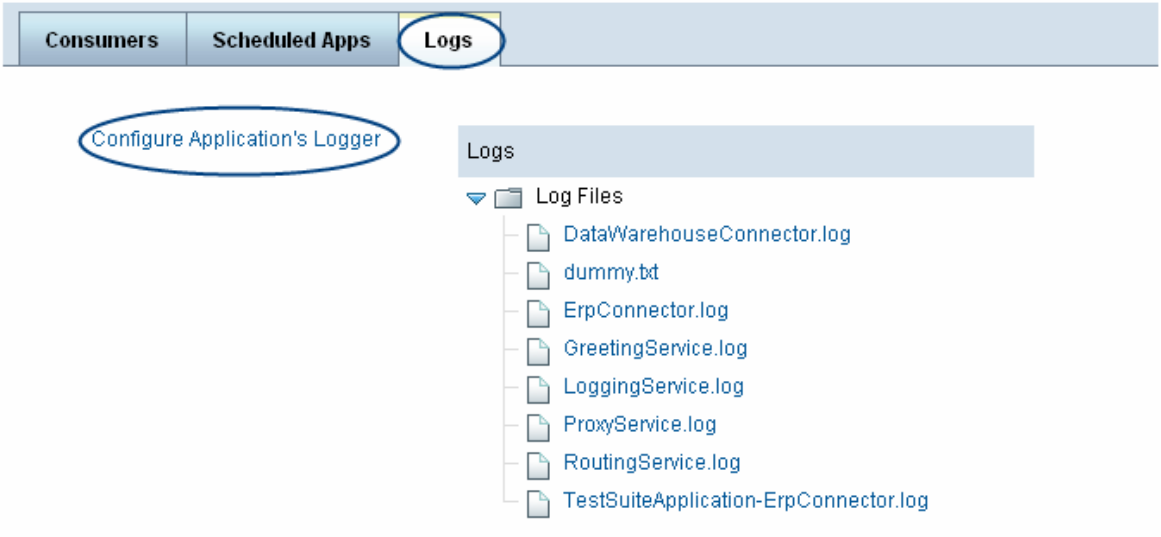
In addition to consumers, a list of Scheduled Applications may also be presented on this page on the `Scheduled Apps` tab. This tab presents a list of any Scheduled Applications associated to the primary application that may be managed similarly as the consumers listed above.



The user may perform the following actions on the Scheduled Applications listed:

- **Edit the scheduled application foundation configuration** by clicking on the "Edit…" button. When the user clicks on the Edit… button, they will be taken to the `Scheduled App Details Page` (see Scheduled Application Details section below) where they can change specific information related to that scheduled application itself. The user may also double click on a scheduled application from the list to be taken to the `Scheduled Application Details Page`.
- **Create a new scheduled application from scratch** by clicking on the "New…" button which will walk the user through a "wizard" for creating a new scheduled application.
- **Delete the selected scheduled application** by clicking on the "Delete…" button.
- **Create a copy of an existing scheduled application** that looks exactly like the selected scheduled app by selecting the "Copy…" button. When a scheduled app is copied, the user will be taken to the `Scheduled Application Details Page` where they can change the scheduled app's name (required) and other configuration information as they see fit.

Finally, the user can select the `Logs` tab where they will be presented with a link to configure the log associated to this application and a list of all known log files that are being used by this and other applications. If the user follows the "Configure Application's Logger" link, they will be taken to the `Logger Configuration page` where they can control the behavior of the Log4J logger associated to this application. If they select one of the log files in the list of known logs, they will be able to view that log.



Below is an overview of some of the new features associated to the Application Details page:

- **Deployment Status -**
- **Revision History**
  - The user is taken to a page that displays a table of the complete revision history for the selected application. This is useful for keeping track of who makes changes to the applications managed by this instance of the ESB.
- **View/Edit runtime environment configuration**
  - The JVM Process Properties page should open. Everything should be blank except for the Process Identifier and Console Identifier text fields. Those fields should be disabled.
- **View most recently used runtime environment**
  - The Recently Used Runtime Environment page should open. There should be six categories of runtime information displayed:
    - JVM Arguments
    - Application Id
    - Application Runner
    - Java Home
    - Classpath
    - Config element file spec
  - All items should be collapsed initially.
- **View most recent test suite summary**

  - The TestStepSummary tree folder should expand and you should see the following child nodes:
    - TotalSteps - Number of steps in the test suite
    - PassedSteps - Number of steps that executed successfully in the test suite
    - FailedSteps - Number of steps that executed unsuccessfully in the test suite
  - Click on the View Entire Summary Document link to see the entire Test Suite Summary document. The entire Test Suite Summary document for the last run of the test suite will open in a new browser window.

- **Search and Replace**
  - Opens the Search and Replace page. Use this interface to perform mass "search and replaces" in the configurations of applications being managed by this ESB.
  - IMPORTANT: This action should be performed with extreme caution

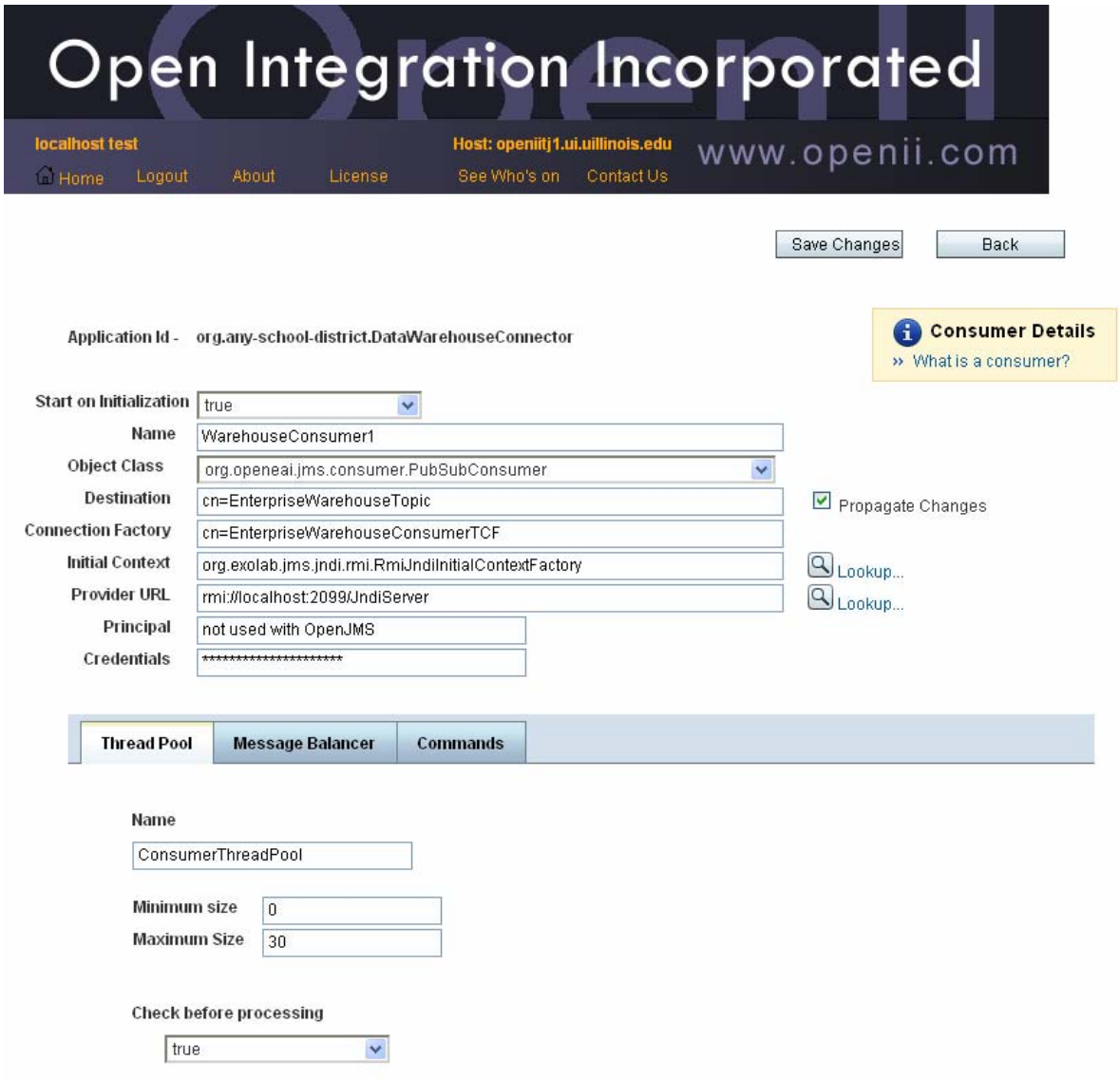
- **Export/Edit XML Configuration**
  - The Export/Edit Application Configuration page should open. The text area in the middle of the page should be populated with XML content that is the configuration for this application.
  - Warning: Changes made on this screen should be made with caution. Click the Edit button to manually edit this application's configuration. Click the Save button to save the changes made to the configuration.

- **Lookup Consumer**
  - When the user clicks on the “Lookup…” button, they will be taken to the `Lookup Component page.` The user can then select from the dropdown list of consumers used by other applications to either Copy or Edit.

Enhanced

The Consumer Details Page

When a user selects a specific consumer on the `Application Details Page` (and on other pages) and chooses to edit that consumer, they will be taken to this page. The `Consumer Details Page` provides an interface for modifying core pieces of information related to a consumer. Currently, this configuration information is specific to JMS. As a result, when configuring an OpenEAI-based consumer, the user will specify settings for features such as the queue to which the consumer will connect, the `ConnectionFactory` that will be used to connect to the broker, and other similar JMS-specific items.



Below is a list of all items that may be specified for a consumer related to how the consumer connects to the messaging infrastructure (broker):

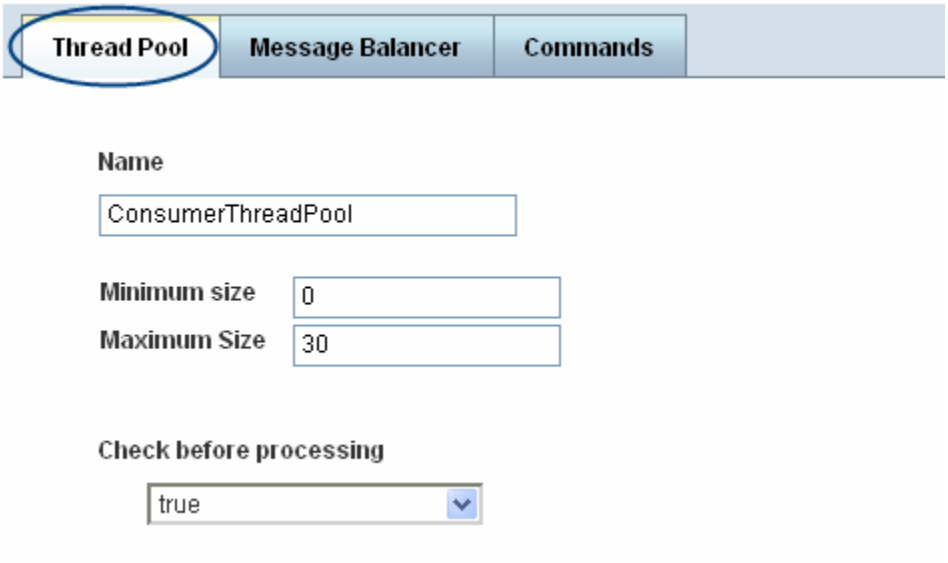
- ***Name:*** This is the name by which this consumer is known in the application's `AppConfig` object. When a developer or an application wants to retrieve consumer from the `AppConfig` object, it may do so by specifying this name in the `getObject(String name)` method of the `AppConfig` object. This name should be as descriptive as possible so developers and administrators can use the name to understand what the consumer means to the gateway being developed. For more details regarding the `AppConfig` object, refer to the OpenEAI API Introduction Document.
- ***Destination*:** This is the name of the JMS queue or topic from which this consumer will consume messages. This is also an "administered object" stored in the server or other JNDI store whose location is specified in "Provider URL" (see below).
- ***Connection Factory Name*:** This specifies the lookup name for the JMS connection factory that will be used by this consumer. This can be either a `TopicConnectionFactory` or a `QueueConnectionFactory`, depending on the type of consumer being configured. A JMS connection factory contains JMS provider-specific about how to connect to the JMS provider. Connection factories are "administered objects" and are stored in a directory server or other store whose location is specified in "Provider URL" (see below). By using connection factories and JNDI, the connection behavior for a consumer is not vendor-specific, which enables organizations to switch JMS providers relatively easily.
- ***Initial Context Factory*:** This is the fully-qualified name of the class the consumer will use to obtain an initial context with the directory server or JNDI store where the administered objects reside. When an organization uses an LDAP repository to store JMS administered objects, this will almost always be `com.sun.jndi.ldap.LdapCtxFactory`. This class name will be based on the repository being used to store JMS administered objects.
- ***Provider URL*:** This is the location in the directory server or other JNDI store where the factories and destinations (JMS administered objects) reside. Examples include: `rmi://localhost:1099/JndiServer` and `ldaps://localhost:636/ou=PointToPoint,ou=Dev,ou=AdministeredObjects`
- ***Security Principal*:** This value specifies the distinguished name of the directory user allowed access the administered objects in the directory server or other JNDI store at the location specified by Provider URL. Example: `uid=Gateway1,ou=Dev,ou=Users`. Note: Security Principal is not used by all JNDI stores.
- ***Security Credentials*:** Specifies the password associated with the Security Principal.
- ***Object Class*:** Specifies the type of consumer that should be instantiated when the gateway is started. This should be the fully-qualified class name of object. Currently, this should only be one of the following classes: `org.openeai.jms.consumer.PointToPointConsumer` for point-to-point (request/reply) message consumption, and `org.openeai.jms.consumer.PubSubConsumer` for publish/subscribe (synchronization) message consumption.
- ***Generic Response*: [todo - need to add details and get it into the Console]**.

Below is an overview of the new features associated to the Consumer Details page:

- **Propagate Changes**
  - Click on the checkbox to automatically propagate destination name changes to other producers in other applications.
- **Initial Context lookup**

  - When the user clicks on the "Lookup…" link, they will be taken to the `Lookup Component page`. The user can then select from the dropdown list of InitialContextFactory values used by other applications. The user can select a value that they would like to apply to the Consumer and click on the Apply Changes button to apply the changes.

- **Provider URL lookup**
  - When the user clicks on the "Lookup…" link, they will be taken to the `Lookup Component page`. The user can then select from the dropdown list of ProviderUrl values used by other applications. The user can select a value that they would like to apply to the Consumer and click on the Apply Changes button to apply the changes.



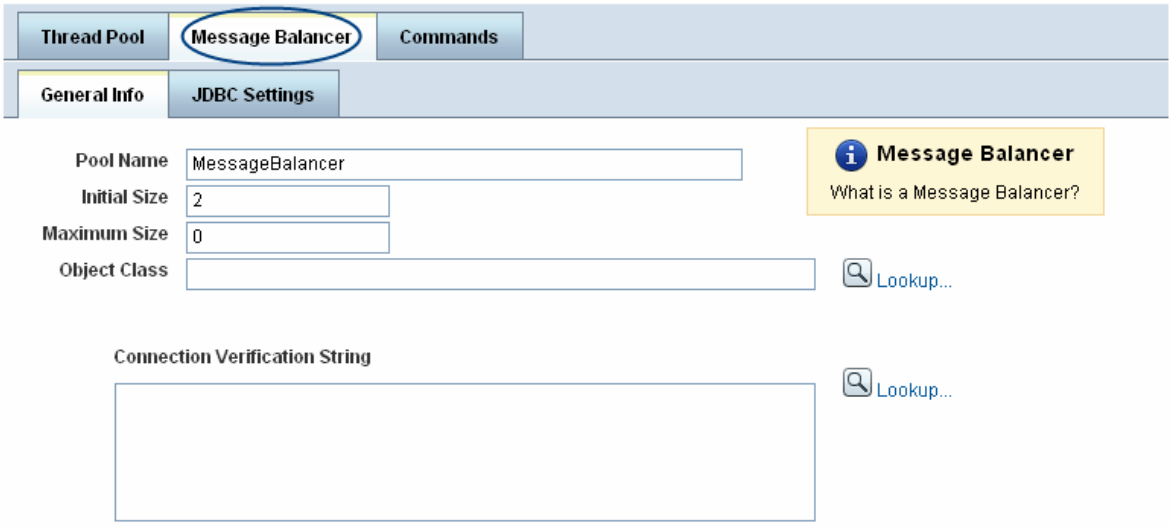
- ***Thread Pool*:** This is the main thread pool in which business logic is executed via "Commands" (see below). When the consumer consumes a message, it will pass the content of the message to the appropriate command and the command will execute the business logic associated to that message. Since a thread pool is used, the `onMessage` method in the consumer's `MessageListener` does not have to wait for the command to complete before consuming the next message. These thread pools should be configured to check their status prior to adding the command execution as a job to the pool by setting "Check Before Processing" to "true". In this case, the consumer will make sure the maximum number of threads is not in use before attempting to execute the command. If the thread pool is busy (maximum number of threads are already in use), it will stop consuming messages and wait until the thread pool can accept more command execution transactions. This reduces the risk of command executions being missed if the gateway terminates abnormally.

- ***Name*:** This is the name of the thread pool.
- ***Maximum Threads*:** This is the maximum number of threads that will ever be in progress at the same time.
- ***Minimum Threads*:** The minimum number of threads that will be allocated to the thread pool.
- ***Check Before Processing*: (true or false).** Checking this check box (true) tells the consumer to make sure there is at least one available "slot" in the thread pool before adding another command execution to the pool.

  If "Check Before Processing" is set to "true" the consumer will wait until there is at least one available "slot" in the thread pool. If it is set to "false" the consumer will continue to add command executions to the thread pool but they will not be processed until a "slot" becomes available. If command executions are continually added to the pool that aren't being processed (because the pool is busy) and the application for some reason crashes, all of those command executions and corresponding consumed messages will be lost. Therefore, the "Check Before Processing" flag can be valuable in controlling how many command executions are in memory and potentially at risk at a given time.

- ***Message Balancer*:** The database connection pool is used by the `MessageBalancer` in `PubSubConsumers` that consume messages off of JMS Topics to ensure that only one consumer per publish/subscribe destination actually processes a message. The `MessageBalancer` is an infrastructure component that `PubSubConsumers` use to determine if another consumer is already processing the message. If so, the consumer will not process the message. Currently, the message balancer uses a database to store the `MessageId` of the message that is consumed to make this determination. This database connection pool is a pool of connections to that repository.

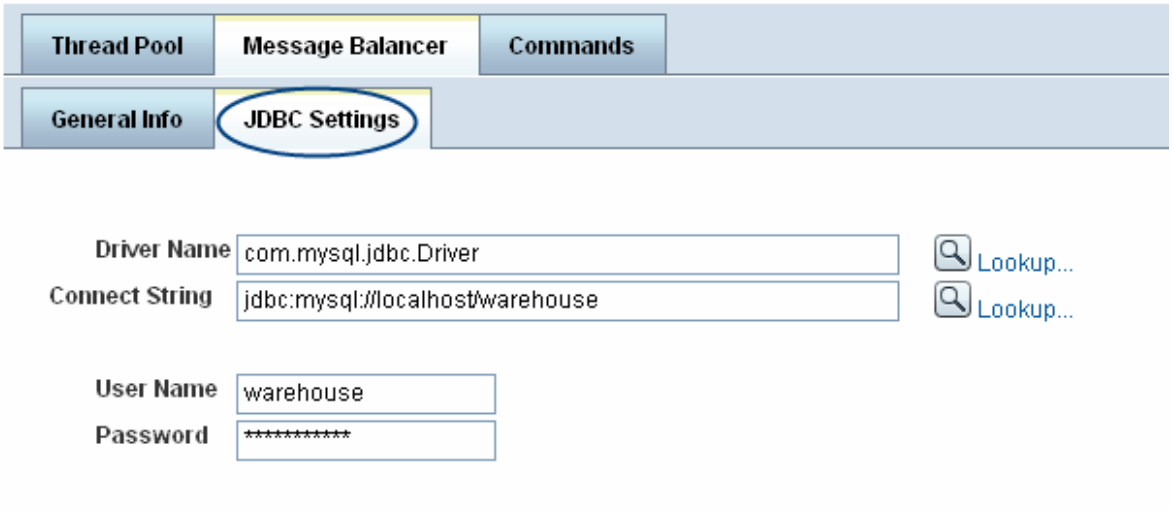
  Note on database pools: For gateways that consume Request-Reply messages (i.e., `PointToPointConsumers` that consume messages from JMS queues) a `MessageBalancer` is not necessary, because when a consumer consumes a message from a queue, that message is no longer available to any other instance that may be connected to that queue. In other words, the first consumer that gets the message processes it. If a database connection pool is not specified (for pub/sub consumers), the consumer will log a warning message during initialization, and all messages consumed by the consumer not using a database connection pool will be processed. If an organization runs multiple instances of gateways that consume sync messages (which should be the case to ensure high availability) this will lead to issues because multiple consumers will be consuming and processing the same messages. In essence, the same messages will be processed more than once by different instances of the gateway. Obviously, this is not desirable. For example, in the case of sync messages, multiple instances of the same gateway might be instructed to create the same record. This will lead to, depending on the underlying data structure into which the data is being inserted, unnecessary duplicate key violation errors or, even worse, duplicate data being stored in the application consuming the sync messages.



The following information may be specified on the ***Message Balancer->General*** Info sub-tab:

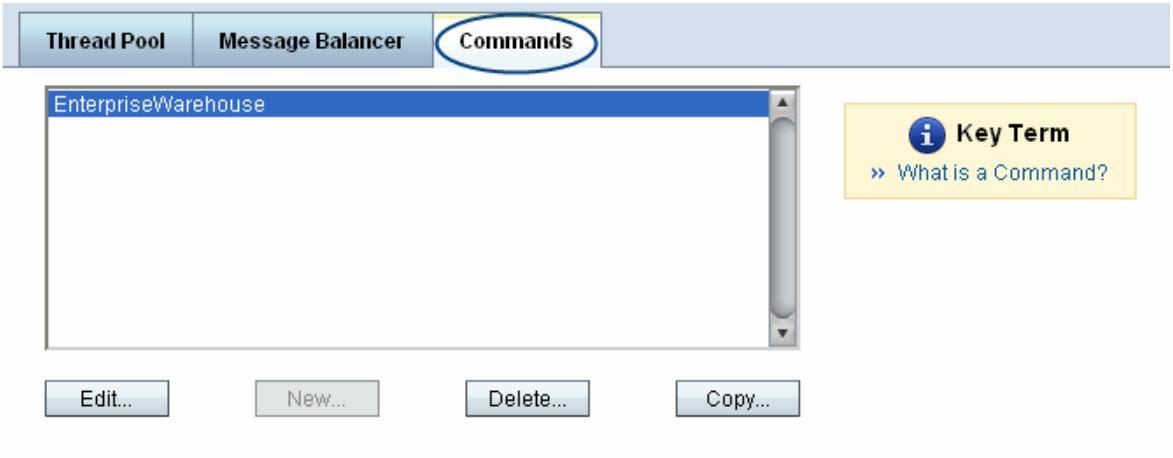
- ***Pool Name*****:** The name of the pool. This is the name by which applications may retrieve the pool from the `AppConfig` object associated to the application or command.
- ***Initial Pool Size*****:** Integer value specifying the number of connections to allocate to the pool initially.
- ***Maximum Pool Size*****:** If specified, this is the maximum number of connections to allocate to the pool. In this case, the pool will only create additional connections if needed until this maximum number is reached. If it is not specified and additional connections are needed, the pool will create connections indiscriminately.
- ***Object Class*****:** The fully-qualified class name of the connection pool to instantiate. For example, if an organization is using the OpenEAI database connection pool, this value would be `org.openeai.dbpool.EnterpriseConnectionPool`.
- ***Verification String*****:** A string used to externalize the verification of database connections as they are retrieved from the pool to ensure that a reliable connection is being returned. There have been cases where not all JDBC implementations support the `isClosed()` method in the JDBC specification. So, this is not always a reliable way to determine if a connection is "good." Users can use this string to provide a **simple** and **efficient** alternative to the `isClosed()` method to verify connection status. Note: having a "simple" and "efficient" alternative is essential here because if a verification string is specified, the pool will attempt to execute that statement each time a connection is returned from the pool to verify its status.

The following information may be specified on the ***Message Balancer->JDBC Settings*** sub-tab:



- ***Driver Name*:** The name of the JDBC implementation to use for the connections in this pool. For example: `oracle.jdbc.driver.OracleDriver`.
- ***Connect String*:** The name of the target database to which the connections in this pool should connect. For example: `jdbc:oracle:thin:@localhost:1521:DB_NAME`. This will be a provider-specific value.
- ***User Name*:** The name of the user to use to connect to the database.
- ***Password*:** The password associated to the user.

The above information relates to how the consumer operates (i.e., how it connects to the JMS provider, how its thread pool works, and what database it uses for its message balancer if it is a `PubSubConsumer`). To instruct the consumer regarding what business logic needs to be executed when a message is delivered to the queue or topic to which it is connected, the user must associate at least one command to that consumer. Commands are Java classes that implement the business logic which will be executed when the consumer consumes a message. So, on the `Consumer Details Page`, the user is presented with a list of commands currently associated to the selected consumer.



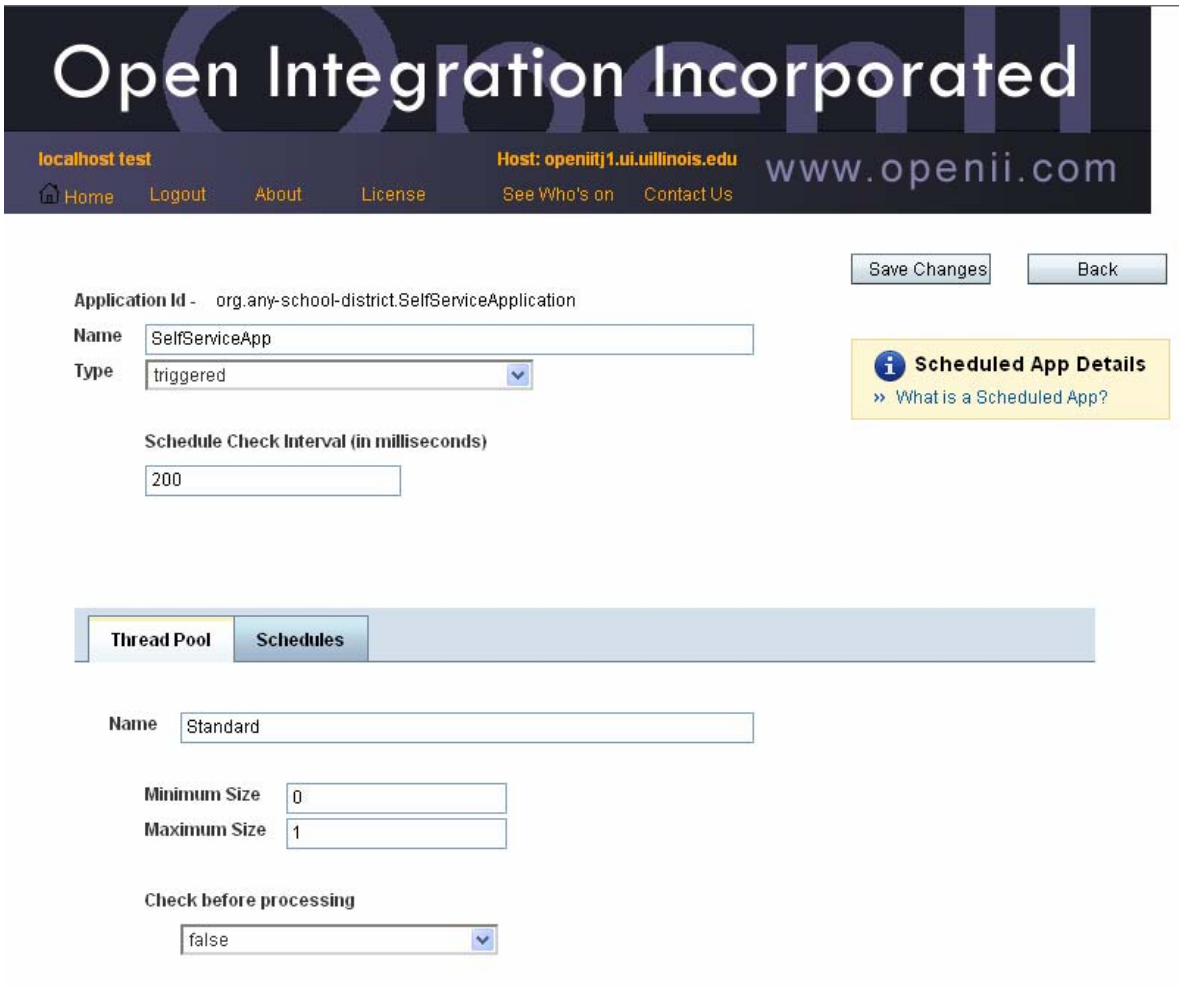
The user may select a specific command and perform several actions associated to it:

- **Create a copy of an existing command** using the selected command as a template by clicking on the "Copy" button. This will add a new command that inherits the same configuration as the selected command to the consumer. The consumer can then be instructed when to execute this new command.
- **Delete the selected command** by clicking on the "Delete" button.
- **View and modify all of the other configurable components used by the command** by clicking on the "Edit Components" button. When this button is clicked, the user will be taken to the `Command Details Page` (see below). On the `Command Details Page` the user can configure components used by the selected command which are required in order for the command to successfully execute the business logic it is implementing.

The Scheduled Application Details Page

When a user selects a specific scheduled application on the `Application Details Page` (and on other pages) and chooses to edit that scheduled application, they will be taken to this page. The `Scheduled Application Details Page` provides an interface for modifying core pieces of information related to a scheduled application. This information is used to control how the scheduled application behaves, i.e., how it executes the business logic assigned to it via Commands.

If the selected scheduled application is an instance of the Test Suite application (see below), a link will be visible that can be used to view the results of previous test suite runs. **[TODO – This needs to be re-positioned up to the Application Details Page discussion.]**



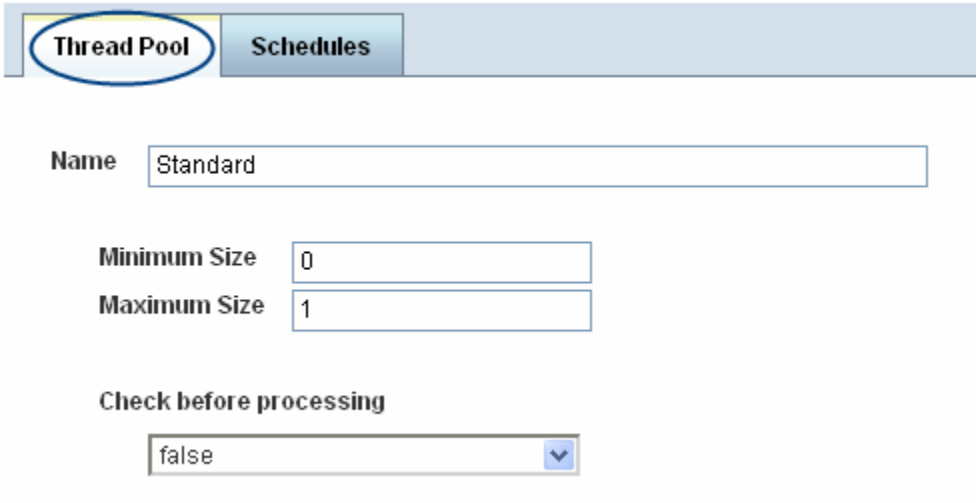
Below is a list of all items displayed on this screen and a description regarding how those items affect the behavior of the scheduled application:

- ***Name***: a name given to this scheduled application for identification purposes.
- ***Type***: the type of scheduled application. There are currently three types of scheduled applications:
    a. Application – is a type of scheduled application that will start, execute the business logic associated to it (via Commands) and exit. This is a fairly typical type of application used many times for batch processing. e.g., it's a "one-time" run of the application.
    b. Daemon – is a type of scheduled application that starts and then executes the business logic on a given schedule (time interval or specific days and times of day). This business logic is executed each time the schedule is met. So, for example, if an organization wants an application that polls a database for changes every 30 seconds they would use a "daemon" scheduled application with a Schedule Check Interval of 30000. This would instruct the scheduled application foundation to execute the business logic (the Command) every 30 seconds.

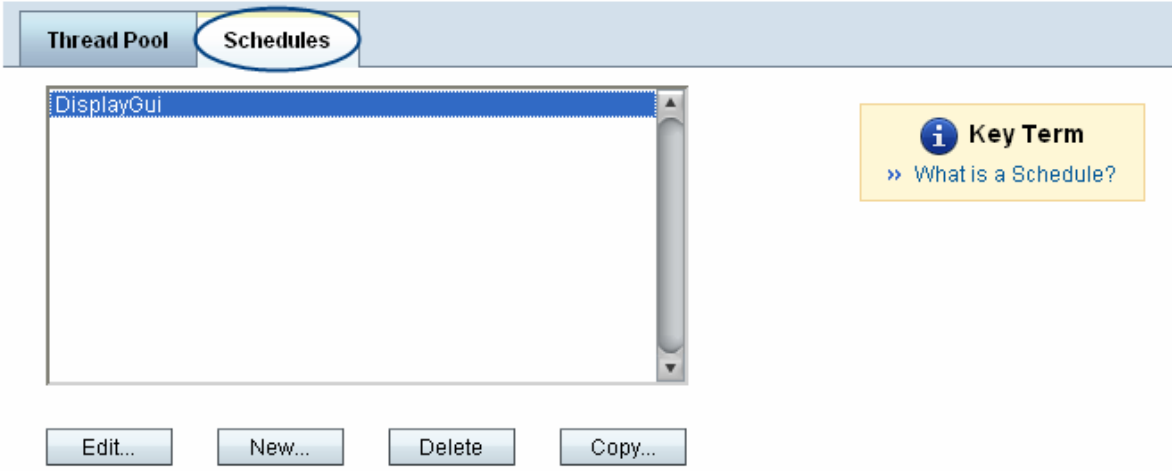
c. Triggered – is a type of scheduled application that starts, executes the business logic and then waits to be triggered to exit. This is useful for applications where the user only wants the business logic to execute once and wait to exit.

- ***Schedule Check Interval***: This value tells the scheduled application foundation how often to check to see if the business logic should be executed. This is especially useful for "daemon" type scheduled applications where the user wants the business logic to be executed on a given interval. i.e., no complex schedule needs to be associated to the business logic. This is one of the most typical methods of specifying how often business logic should be executed.

- ***Thread Pool tab***. Just like consumers, scheduled applications execute their business logic (commands) in threads. This is the thread pool in which that business logic will be executed when the schedule is met. The configuration of this thread pool is exactly the same as for consumers listed above.



- ***Schedule Tab***. The scheduled application foundation uses these schedules to determine when business logic should be invoked. As mentioned above, the most common type of schedule is configured via the Schedule Check Interval. However, Schedules can be used to allow more flexibility in specifying when business logic should be executed. This version of the console does not support this advanced feature but it will in the future. The user can perform the following actions on the list of schedules associated to the selected scheduled application:
  a. Select a schedule from the list and click on the Edit… button to be taken to the Schedule Details page (see below) where they can specify additional configuration information for the schedule. Double clicking on a schedule will also navigate the user to the Schedule Details Page.
  b. Delete a schedule from the list by selecting the schedule and clicking on the Delete button.
  c. Copy an existing schedule by selecting a schedule and clicking on the Copy… button. This will create a copy of the selected schedule and take the user to the Schedule Details page where the user can modify information about the new schedule that was just copied.

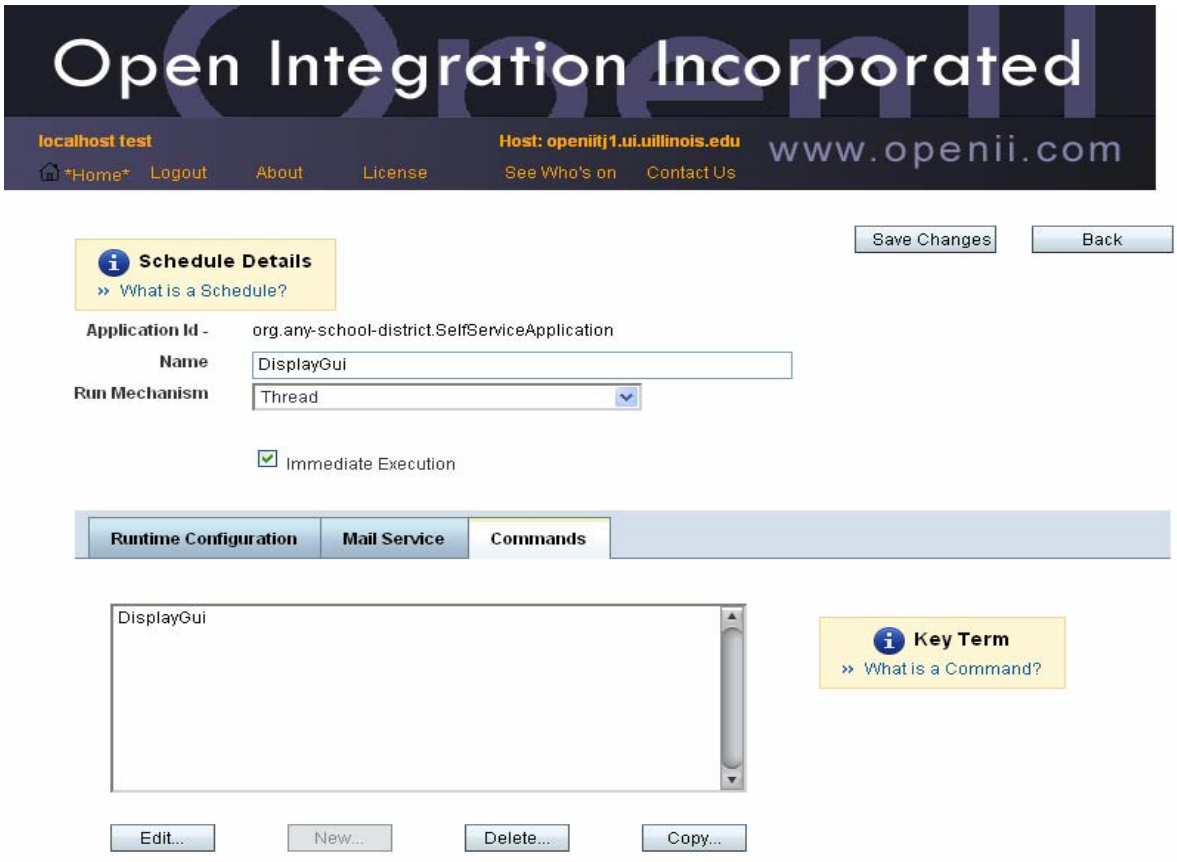


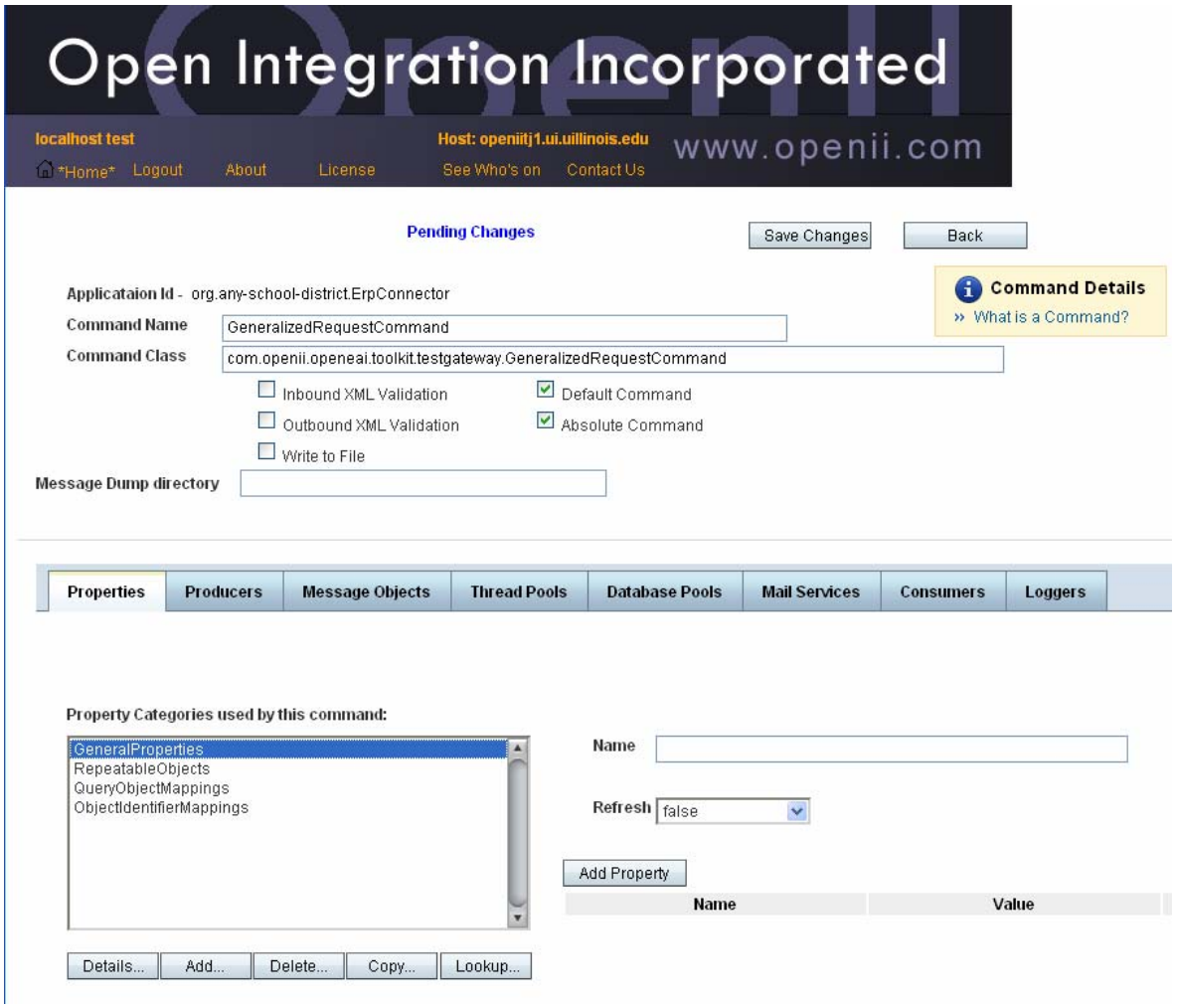
The Schedule Details Page

The Schedule Details Page allows the user to specify information relating to the configuration of the Schedule associated to the selected Scheduled Application. Schedules are used to control when business logic is executed using the OpenEAI Scheduled Application foundation. When a Schedule is "met" all commands associated to that Schedule are executed. So, if a Schedule is set for Monday, Wednesday and Friday at 1pm all Commands associated to that schedule will be executed and their business logic performed. As mentioned above, this advanced feature of Schedule configuration is not included in this release of the Console but it will be in the future. So, this page exists to allow that future enhancement as well as to provide the list of Commands that will be executed when the schedule is met using the less advanced scheduling mechanisms (Schedule Check Interval).

On this page, the user is presented with some general information regarding the selected schedule. They may change the name of the schedule (for identification purposes) and they may specify that the schedule should execute "immediately." Schedules that are configured to execute immediately ignore any other scheduling configuration and simply execute the business logic associated to them (the Commands).

The user is also presented with a list of all Commands associated to the schedule and they may perform the same type of functions on these commands as they can on the Consumer Details page mentioned above. Most often, the goal will be to edit the command in some way to control what components are available to that command at runtime when it is performing the business logic associated to it (see below).


Open Integration Incorporated
localhost test
Host: openiitj1.ui.uillinois.edu
www.openii.com
Home Logout About License See Who's on Contact Us
Save Changes
Back
Schedule Details
» What is a Schedule?
Application Id - org.any-school-district.SelfServiceApplication
Name DisplayGui
Run Mechanism Thread
Immediate Execution
Runtime Configuration
Mail Service
Commands
DisplayGui
Key Term
» What is a Command?
Edit...
New...
Delete...
Copy...



Enhanced

The Command Details Page

When the user selects a command on the Consumer Details Page or on the Schedule Details Page and clicks the "Edit" button, they are taken to the Command Details Page. This page provides the ability to modify and save command-specific configuration information. It also lists all lower-level components related to the selected command. This includes Producers, Thread Pools, Database Pools, Properties, Mail Services, Message Objects, etc. These are components that the command uses when executing the business logic associated to it. The number and type of components required depend solely on the business logic being executed by the command (i.e., the requirements that are implemented by the command).

The user can select any of these components related to the command and view or modify the configuration related to that component. For each list of components associated to a command, there will be an opportunity for the user to select that component and modify details specifically for that type of component. These components are the items used by the command when executing the business logic associated to the command. For example, if the command is passed a BasicPerson-Create-Sync message it might populate a BasicPerson Java object and store that person information in a database. In this example, the command would have a BasicPerson Java

object associated to it as well as a database connection pool that it uses to insert the record into the database.

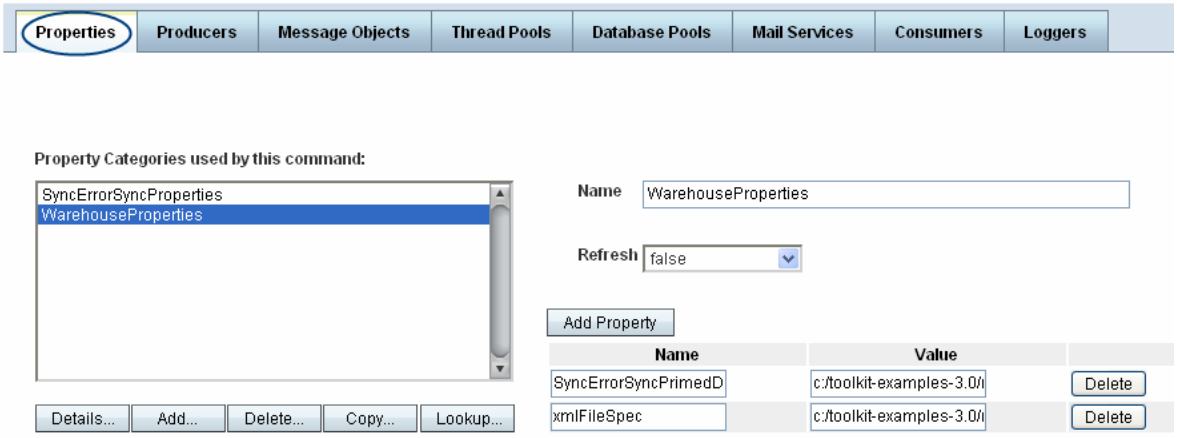
On the `Command Details Page`, the user is presented with a tab set that that lists all the components associated to the command. When the user clicks on a particular tab, the related details are shown that allows the user to configure information specifically related to the component. The user can then make changes, additions, and deletions for that particular item and save it back to the configuration for the command being maintained.

When changes are made to any of the components used by the command, the "Save Changes" button will save the changes back to the configuration document associated to the selected application.

The components the user can manage via the `Command Details Page` include:

Properties

When the user clicks on the "Properties" tab, a list of existing properties associated to the selected command is shown. The items shown in the list are really categories of properties. It is often useful to categorize properties to make the configuration and development of an application more meaningful. By providing a "container" that is named along with a group of name/value pairs within that container, the developer can retrieve sets of properties by name for use in the application.



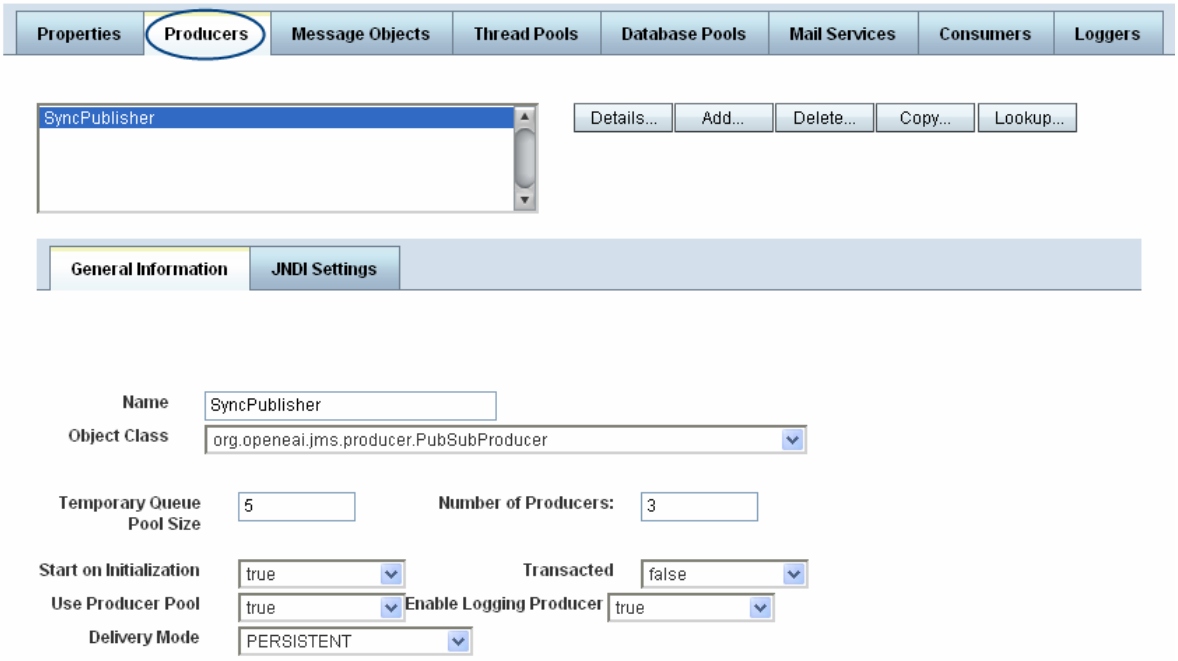
The user may perform the following actions on the list of property categories:

- **Select a property category and view its properties** by clicking the "Details" button. Then, the user can make changes to the content of the properties object (i.e., change property names and/or values, add new name/value pairs to the properties object, delete existing name/value pairs).
- **Add a new named property category and then add name/value pairs to that category** by clicking the "Add…" button below the list of property categories. This will create a new category of properties that can be filled with whatever name/value pairs that is appropriate for the application/command being developed. The properties that are required, will vary based on the design/implementation of the command being developed.

- **Select a property category and delete that object along with all the name/value pairs associated to it.** When you do this, it removes the category of properties from the list of available property categories available to the command. If the command refers to the category that you delete, it may have errors at runtime so deleting property categories should be done in coordination with the development of the application/command.
- **Select a property category and copy it to a new name.** This is basically the same as creating a new category of properties except that by doing this, existing name/value pairs get copied over from the selected property category and is one way in which the command may retrieve the property category from its `AppConfig`.
- **Lookup property categories** by clicking on the "Lookup" button. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Component page.` The user can then select from the dropdown list of property categories used by other applications to either Copy or Edit.

Producers

When the user clicks on the "Producers" tab, a list of existing producers associated to the selected command is shown. Producers are used by commands to publish synchronization messages to topics or send requests to queues. Producers are OpenEAI-based components that are used to transport the XML enterprise messages to an end point. A command may have as many producers as it needs to execute its business logic. Producers are configured very similar to consumers since they are also basic JMS foundation components.

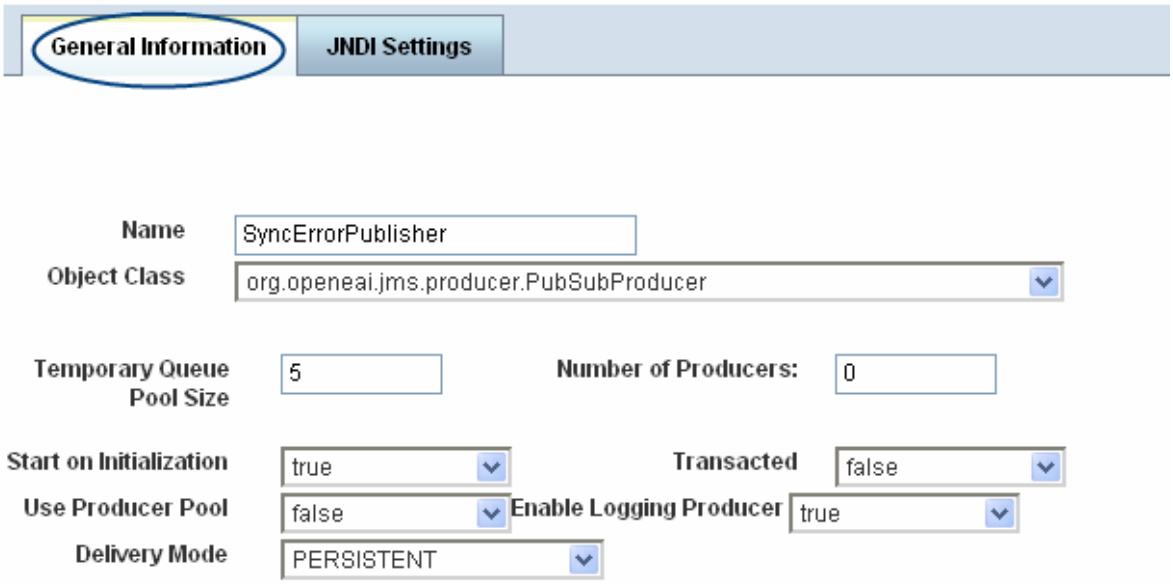


The user may perform the following actions on the list of producers:

- **Select a producer and view/modify its details** by clicking the "Details" button. Then, they can make changes to the configuration of the producer object (i.e., change the destination, etc.).
- **Add a new named producer** and then configure it appropriately.
- **Select a producer and delete it.** When you do this, it removes the producer from the list of available producers available to the command. If the command refers to the producer that you delete, it may have errors at runtime so deleting producers should be done in coordination with the development of the application/command.
- **Select a producer and copy it to a new name.** This is basically the same as creating a new producer except that by doing this, existing configuration information gets copied over from the selected producer. It is often the case that much of the configuration information associated to a producer is the same with the exception of the destination name so this can be a valuable time saver.
- **Lookup producers** by clicking on the "Lookup" button. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Component page.` The user can

then select from the dropdown list of producers used by other applications to either Copy or Edit.

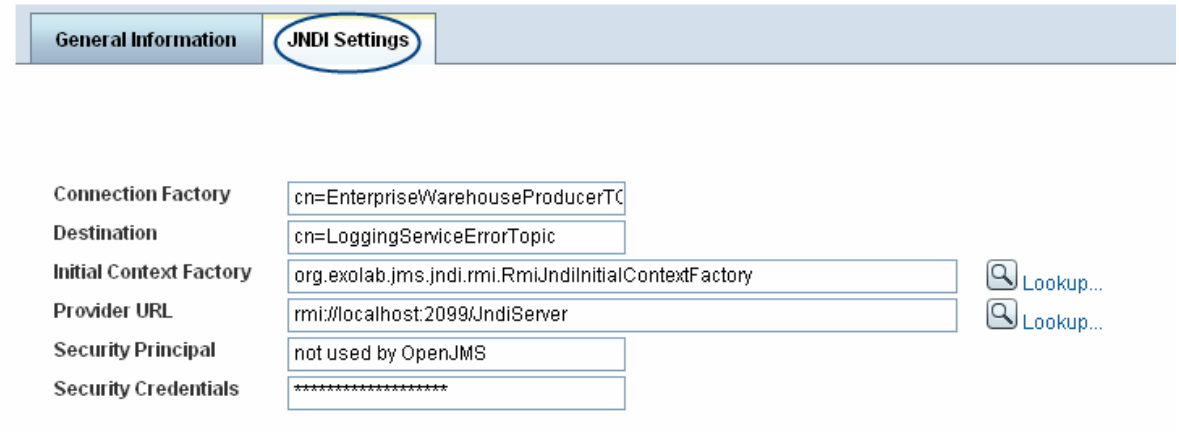
The following information may be specified for a producer on the ***Producers->General Information*** sub-tab:



- ***Name*****:** This is the name by which the command will likely retrieve the producer. Each producer listed in this list will be loaded into the Command's `AppConfig` object and may be retrieved by the command at runtime when appropriate.
- ***Object Class*****:** Specifies the type of producer that should be instantiated when the application/command is started. This should be the fully-qualified class name of object. Currently, this should only be one of the following classes: `org.openeai.jms.consumer.PointToPointProducer` for point-to-point (request/reply) message production and `org.openeai.jms.consumer.PubSubProducer` for publish/subscribe (synchronization) message production.
- ***Temporary Queue Pool Size:*** Indicates the number of temporary queues that should be established for the producer when it initializes itself (i.e., when it starts). The default for this is 5 and it is a technique used by the OpenEAI Producer foundation that reduces the cost of sending a message, much like database connection pools reduce the cost of connecting to a database. The temporary queues are established when the producer is first started so they don't have to be created when a message is actually produced.
- ***Number of Producers:*** Indicates the number of Producers that should be created. If this number is greater than one (1), it would indicate that the developer of the application intends to use an OpenEAI Producer Pool component which again provides a mechanism to reduce the cost of sending messages and potentially increases the overall volume that can be performed by the application.
- ***Start on Initialization (true or false):*** This tells the producer how it is to behave when it configures itself. Generally, this will be true indicating that the producer should not only configure itself but it should actually establish a connection to the broker which means it's ready to produce messages whenever it's needed.

- ***Transacted (true or false):*** This specifies whether the JMS session sending these messages is to be transacted or not (i.e., require a commit before it can actually be consumed).
- ***Use Producer Pool (true or false):*** **TODO**
- ***Enable Logging Producer (true or false):*** Indicates whether or not a PubSub Logging Producer should be used when this producer publishes messages (in the case of PubSubProducers).
- ***Delivery Mode:*** **TODO**

The following information may be specified for a producer on the ***Producers->JNDI Settings*** sub-tab (this information tells the producer how to connect to the JMS provider):



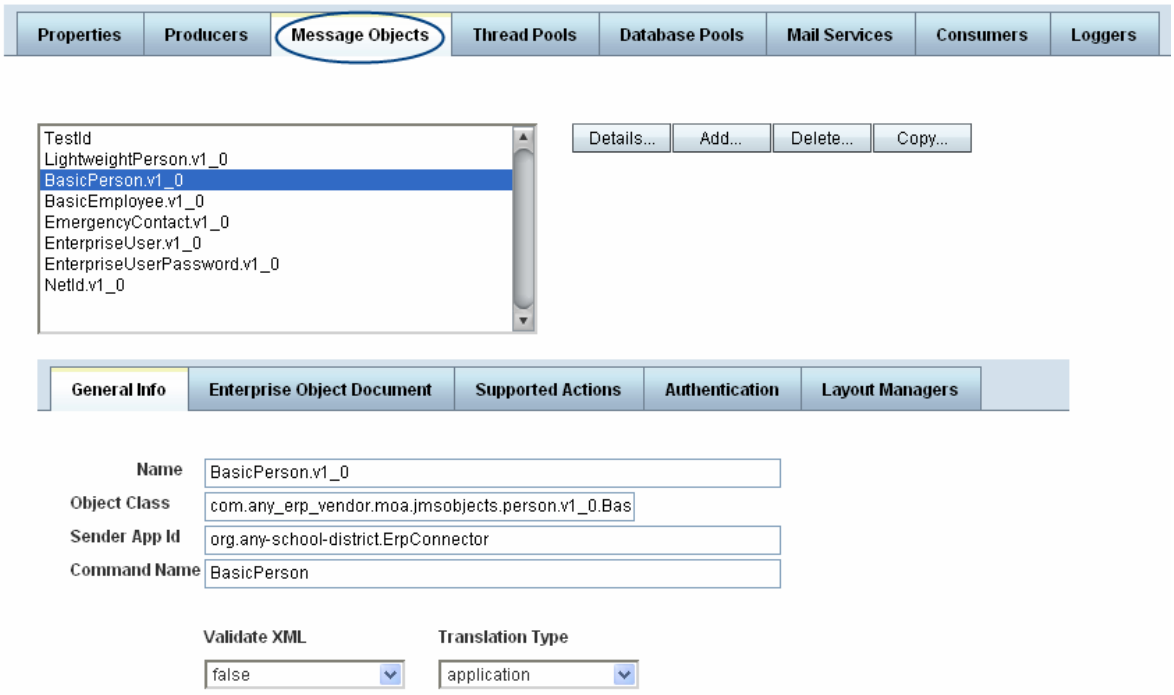
- ***Connection Factory*****:** This specifies the lookup name for the JMS connection factory that will be used by this producer. This can be either a `TopicConnectionFactory` or a, `QueueConnectionFactory` depending on the type of producer being configured. A JMS connection factory contains JMS provider-specific information about how to connect to the JMS provider. Connection factories are "administered objects" and are stored in a directory server or other store whose location is specified in "Provider URL" (see below). By using connection factories and JNDI, the connection behavior for a producer is not vendor specific which will allow an organization to switch JMS providers relatively easily.
- ***Destination*****:** This is the name of the JMS queue or topic to which this producer will send messages. This is also an "administered object" stored in the server or other JNDI store whose location is specified in "Provider URL" (see below).
- ***Initial Context Factory*****:** This is the fully-qualified name of the class the producer will use to obtain an initial context with the directory server or JNDI store where the administered objects reside. When an organization uses an LDAP repository to store JMS administered objects, this will almost always be `com.sun.jndi.ldap.LdapCtxFactory`. This class name will be based on the repository being used to store JMS administered objects.
- ***Provider URL*****:** This is the location in the directory server or other JNDI store where the factories and destinations (JMS administered objects) reside. Examples include: `rmi://localhost:1099/JndiServer` and `ldaps://localhost:636/ou=PointToPoint,ou=Dev,ou=AdministeredObject`
- ***Security Principal:*** This is the directory server user that will be used to lookup the ConnectionFactory and Destination if those administered objects are in fact stored in a directory server.
- ***Security Credentials:*** This is the password associated to the directory server that will be used to lookup the ConnectionFactory and Destination if those administered objects are in fact stored in a directory server.
- ***Lookup Initial Context factory:*** When the user clicks on the "Lookup…" link, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of InitialContextFactory values used by other applications. The user may select a value that they would like to apply to this Producer and click on the Apply Changes button to apply the changes.
- ***Lookup Provider URL:*** When the user clicks on the "Lookup…" link, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of

ProviderUrl values used by other applications. The user may select a value that they would like to apply to this Producer and click on the Apply Changes button to apply the changes.

Message Objects

When the user clicks on the "Message Objects" tab, a list of existing message objects associated to the selected command is shown.

Message objects are the enterprise data, or business objects, used or acted on by an application or command. These are OpenEAI-based Java business objects that correspond to objects defined during integration analysis. These objects are retrieved at runtime and populated with data before performing some action on the object (such as create, delete, or update). Likewise, these objects are populated when a consumer consumes a message and hands that message off to a command for execution. The command often takes the XML message and uses it to populate a message object or objects before calling database procedures or something else that will ultimately persist the data contained within the object. So, in a command, message objects are used extensively, and depending on the business logic being executed by the command, the list of message objects will vary (i.e., it will only be configured to use the objects it needs to perform the business logic at hand).

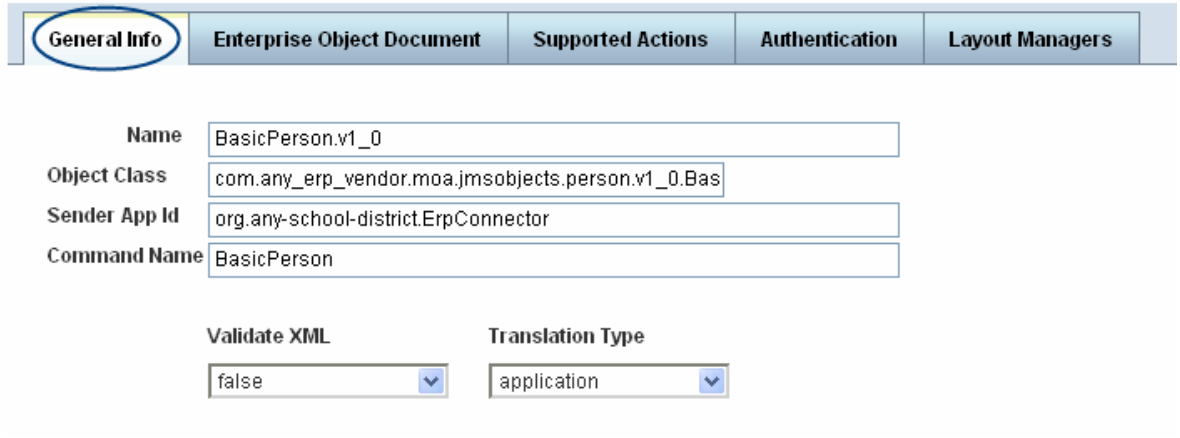


The user may perform the following actions on the list of message objects associated to the command:

- **Select a message object and view/modify its details** by clicking the "Details" button. Then, the user can make changes to the configuration of the message object (i.e., modify the information mentioned above).
- **Add a new named message object** and then configure it appropriately.

- **Select a message object and delete it.** When you do this, it removes the message object from the list of available message objects available to the command. If the command refers to the message object that you delete, it may have errors at runtime so deleting message objects should be done in coordination with the development of the application/command.
- **Select a message object and copy it to a new name.** This is basically the same as creating a message object, however doing it this way permits existing configuration information to be copied from the selected message object. It is often the case that much of the configuration information associated to related message objects is the same with the exception of the name, object class, and primed documents, so this can be a valuable time saver.

The following information may be specified for a selected message object on the ***Message Objects->General Info*** sub-tab:

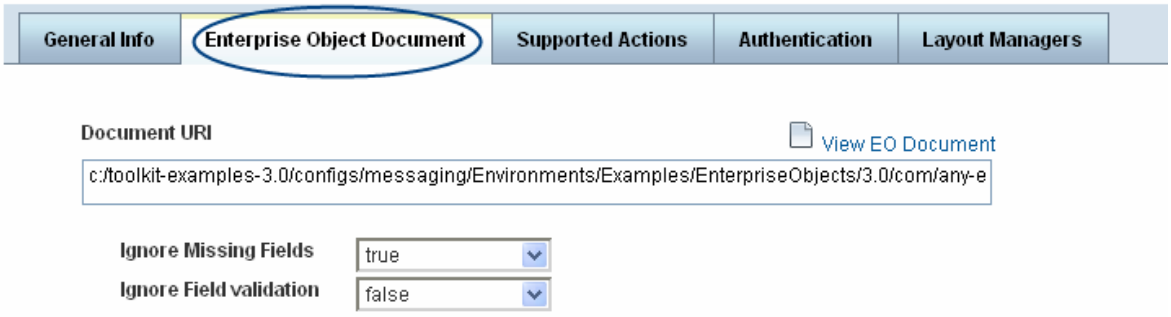


- ***Name*:** The name of the object as it will be known by the application's or command's `AppConfig` object. This is one way in which the application or command can retrieve the object from `AppConfig`. A good naming convention is: Name.version. For example: `BasicPerson.v1_0`. By following a naming convention similar to this, the application can use some of the meta-data information contained in the message consumed to derive a name and will not have to "hard code" references to named objects.
- ***Object Class*:** The full class name of the object that will be instantiated. For example: `com.foo.moa.jmsobjects.person.v1_0.BasicPerson`. These are objects generated by the Message Object Generation application, so these class names will vary by organization.
- ***Sender App ID*:** This value is used when a message is produced by the message object. It indicates the name of the application sending the message and is used by various infrastructure components (such as Request Proxy and Logging Service). Naming conventions should be adopted and followed consistently. For example, many organizations use reverse domain name conventions for application names as well, so an example might look like this: `com.foo.hrapps.SelfServiceApplication`.
- ***XML Validation* (true or false):** If set to true, this will force XML validation at runtime. When an object, like `BasicPerson` is populated and then an action, like create is invoked, an XML message is built by the underlying OpenEAI foundation. In this example, a `BasicPerson-Create-Request` XML message is built. If XML validation is true, this message will be validated from an XML perspective when that `create` action is invoked.

This means, the underlying foundation will use the XML parser to ensure that the structure of that message matches the definition specified by the constraint (DTD or Schema) referenced by the document. Generally, this should always be set to false in a production environment because XML validation does impose additional resource requirements and could contribute to poor application performance. Ideally, by the time an application gets deployed in a production environment, there should be very little chance that the resulting XML document will be invalid from an XML perspective.

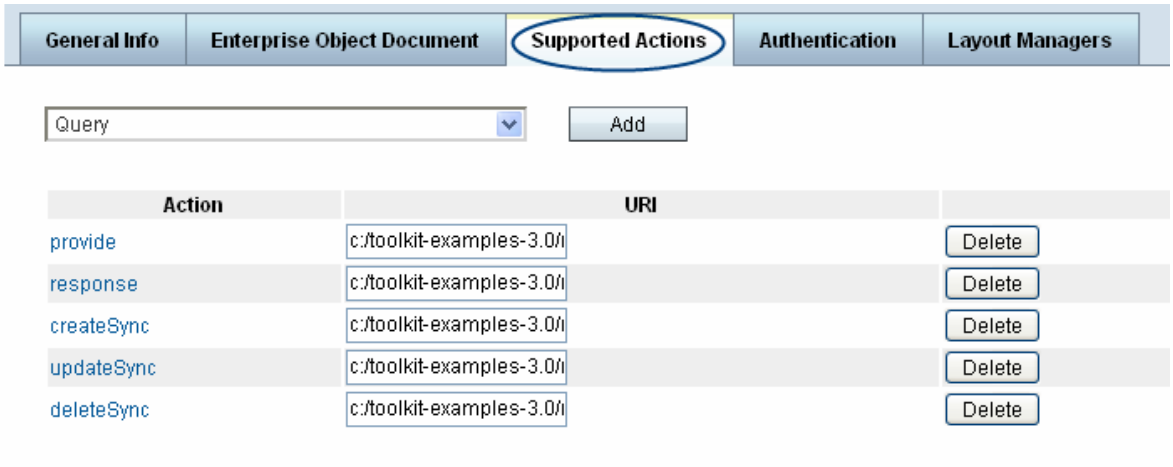
- ***Command Name:*** Is a property associated to the JMS message that is sent on behalf of this message object when one of the OpenEAI actions are performed (Query, Create, Update, Delete etc.). If specified, this value will be used and the consumer of the message will use this command name property to determine which command to execute.

The following information may be specified on the ***Message Objects->Enterprise Object Definition*** sub-tab. ***Enterprise Object Documents*** (EO docs) are used by the message objects for a variety of purposes. They provide general formatting rules (such as length, data types and whether or not a field is required) as well as translations and the ability to "scrub" data being stored in an enterprise object. Every message object that is used by an application (i.e., every object that can be added here) must reference an EO document that describes these rules. The EO docs themselves are also generated by the Message Object Generation application when it generates the Java objects. Once an EO document is generated, analysis dictates the additional information that must be specified in the document. EO documents provide a mechanism to apply enterprise data rules on business objects that goes far beyond the capabilities of XML DTD or Schema which as mentioned above are primarily concerned with structure validation. They provide a mechanism for applying very specific rules to the data that is being put into these business objects.



- ***Document URI*:** The URI of the EO document that will be used by this object. Currently this should be the fully qualified path to the EO document, however in future releases of the Console, much of this will be handled automatically.
- ***Ignore Missing Fields* (true or false):** N/A.
- ***Ignore Validation* (true of false):** When this value is "true" (checked), no formatting, translations, or scrubbing will be performed on the data as it is being put into the business objects via its setter methods. This is completely different from XML validation as it applies to the data being put into the object as opposed to the structure of the XML document that is a result of the object being used in a message that is sent.

The following information may be specified on the ***Message Objects->Supported Actions*** sub-tab:



- ***Primed Documents*:** Primed documents are used by the message object when an action is performed on the object such as `create`, `update`, or `delete`. When the action method is invoked the object uses the data that currently resides within it and places that data into the appropriate XML document. The XML document is then transported to a destination via a JMS Queue or Topic.

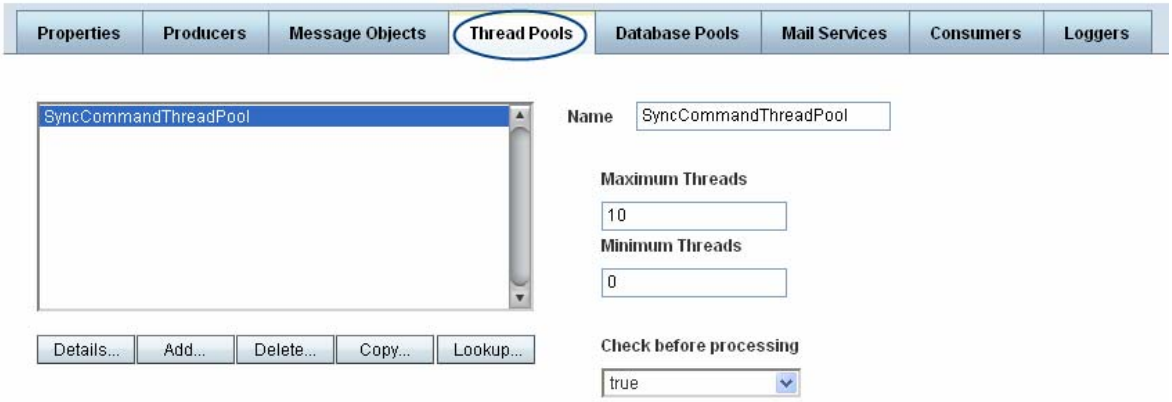
  Primed documents are used as the templates for these documents. They are generally the result of the analysis process employed to determine which message objects need to exist and which applications and/or gateways will need to be developed. The OpenEAI Message Object API foundation uses the primed documents as templates so it doesn't have to build these documents from scratch each time. This results in a significant performance boost because the primed documents are loaded into memory during initialization and the application just modifies the document appropriately at runtime (i.e., it replaces the `DataArea` with the content of the object and sets some `ControlArea` elements to the appropriate values).

  When the user "selects" the actions that will be "performed or supported" by this command on the selected message object, the Console will pre-fill the primed document information associated to that selected action.

- ***Layout Managers*:** XmlLayout is the only layout manager being supported this release.

Thread Pools

When the user clicks on the "Thread Pools" tab, a list of existing thread pool objects associated to the selected command is shown. Thread pools are used by applications to increase the processing capacity of the application while providing the ability to control how many threads the application has at its disposal (similar to database connection pools, below). Since commands can be considered applications in themselves, they often use thread pools as well.



The user may perform the following actions on the list of thread pools associated to the command:

- **Select a thread pool and view/modify its details** by clicking the "Details" button. The user can then make changes to the configuration of the thread pool (i.e., change the maximum pool size, etc.).
- **Add a new named thread pool** and configure it appropriately.
- **Select a thread pool and delete it.** When you do this, it removes the thread pool from the list of available thread pools available to the command. If the command refers to the thread pool that you delete, it may have errors at runtime so deleting thread pools should be done in coordination with the development of the application/command.
- **Select a thread pool and copy it to a new name.** This is basically the same as creating a thread pool, however doing it this way permits existing configuration information to be copied from the selected thread pool.
- **Lookup Thread Pools** by clicking on the "Lookup" button. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Component page.` The user can then select from the dropdown list of Thread Pools used by other applications to either Copy or Edit.

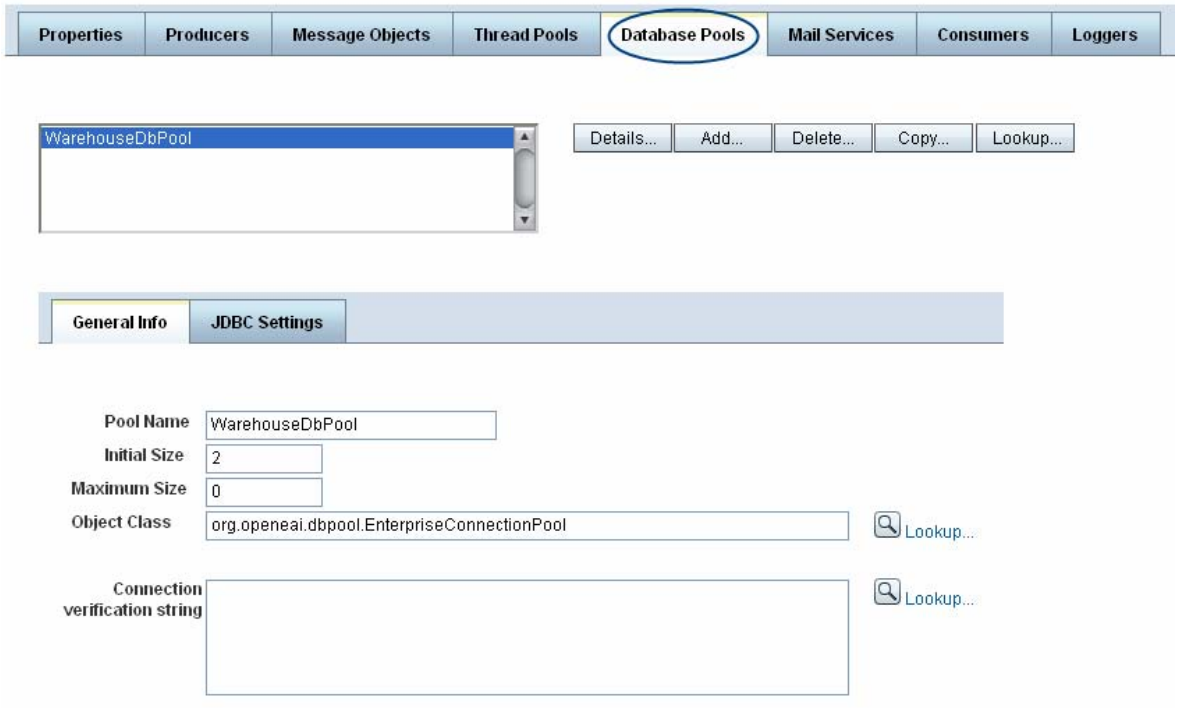
The following information may be specified for a thread pool:

- ***Name*:** This is the name of the thread pool and is one way in which the command may retrieve it from its `AppConfig`.
- ***Maximum Threads*:** This is the maximum number of threads that will ever be in progress at the same time.
- ***Minimum Threads*:** The minimum number of threads that will be allocated to the thread pool.
- ***Check Before Processing* (true or false):** Checking this check box (true) gives the command the opportunity to make sure there is at least one available slot in the thread pool before adding another process execution to the pool.

Enhanced

Database Pools

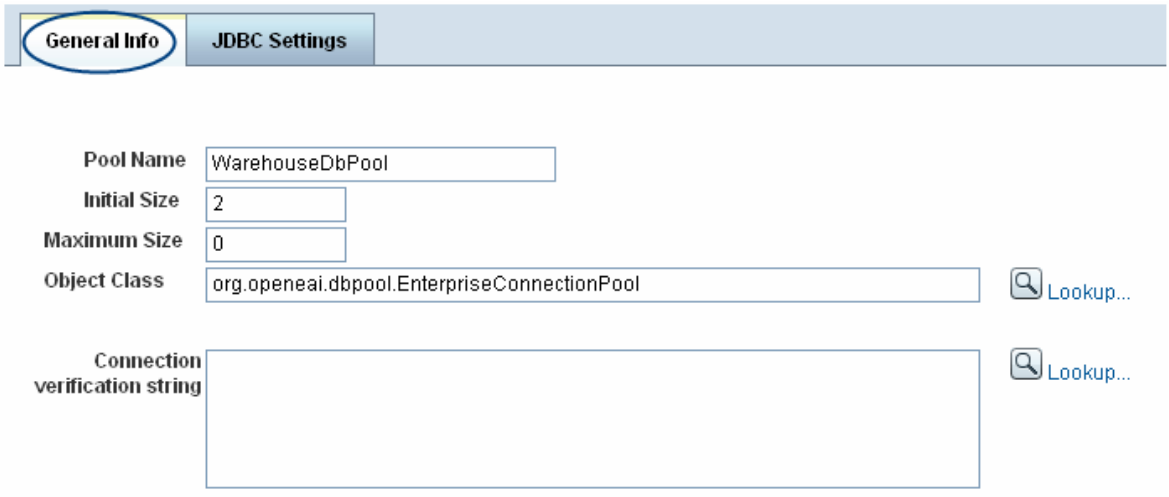
When the user clicks on the "Database Pools" tab, a list of existing database pool objects associated to the selected command is shown. Database connection pools are used in many Java applications to control the number of connections an application makes to a database. Additionally, they are used to reduce or eliminate the need for developers to establish connections to the database when, for example, an insert is to be performed. Instead, by using a database connection pool, the developer has pre-established database connections that are ready to use and the costly overhead associated to establishing a connection is avoided.



The user may perform the following actions on the list of database connection pools associated to the command:

- **Select a pool and view/modify its details** by clicking the "Details" button. The user can then make changes to the configuration of the pool (i.e., change the maximum pool size).
- **Add a new named pool** and then configure it appropriately.
- **Select a pool and delete it.** When you do this, it removes the database pool from the list of available pools available to the command. If the command refers to the database pool that you delete, it may have errors at runtime so deleting database pools should be done in coordination with the development of the application/command.
- **Select a pool and copy it using a new name.** This is basically the same as creating a database connection pool from scratch, except that by doing this, existing configuration information is copied over from the selected database connection pool.
- **Lookup Database Pools** by clicking on the "Lookup" button. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of Database Pools used by other applications to either Copy or Edit.
- **Lookup Object Class** by clicking on the "Lookup" link next to the Object Class text box. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of ObjectClass values used by other applications. The user may select a value that they would like to apply to this Database Pool and click on the Apply Changes button to apply the changes.
- **Lookup Connection verification String** by clicking on the "Lookup" link next to the Connection Verification String text box. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of Verification String values used by other applications. The user may select a value that they would like to apply to this Database Pool and click on the Apply Changes button to apply the changes.

The following information may be specified for a database connection pool on the ***Database Pools->General Info*** sub-tab:

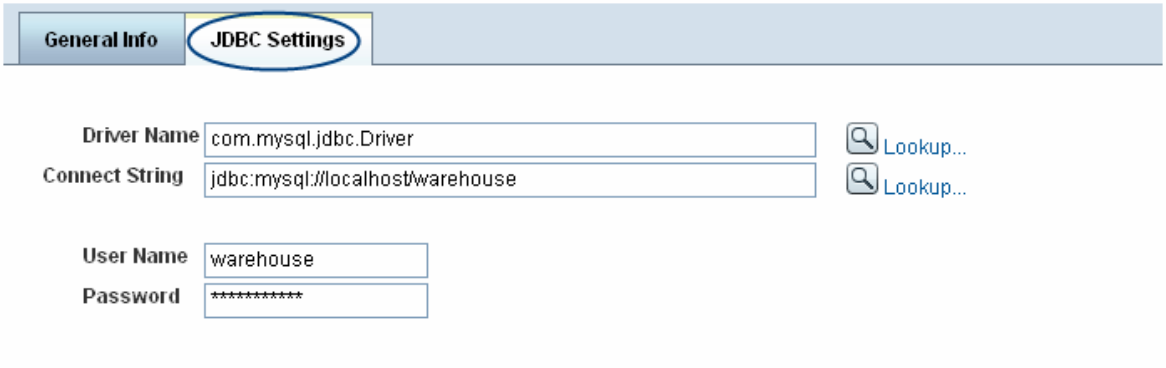


- ***Pool Name*:** The name of the pool. This is the name by which applications may retrieve the pool from the `AppConfig` object associated to the application or command.
- ***Initial Pool Size*:** Integer value specifying the number of connections to allocate to the pool initially.
- ***Maximum Pool Size*:** If specified, this is the maximum number of connections to allocate to the pool. In this case, the pool will only create additional connections if needed until this

maximum number is reached. If it is not specified and additional connections are needed, the pool will create those connections indiscriminately.

- ***Object Class*:** The fully-qualified class name of the connection pool to instantiate. For example, if an organization is using the OpenEAI database connection pool, this value would be `org.openeai.dbpool.EnterpriseConnectionPool`.
- ***Verification String*:** A string used to externalize the verification of database connections as they are retrieved from the pool to ensure that a reliable connection is being returned. There have been cases where not all JDBC implementations support the `isClosed()` method specified by the JDBC specification. So, this is not always a reliable way to determine if a connection is "good." Users can use this string to provide a **simple** and **efficient** alternative to the `isClosed()` method to verify connection status. Note: having a "simple" and "efficient" alternative is essential here because if a verification string is specified, the pool will attempt to execute that statement each time a connection is returned from the pool to verify its status.

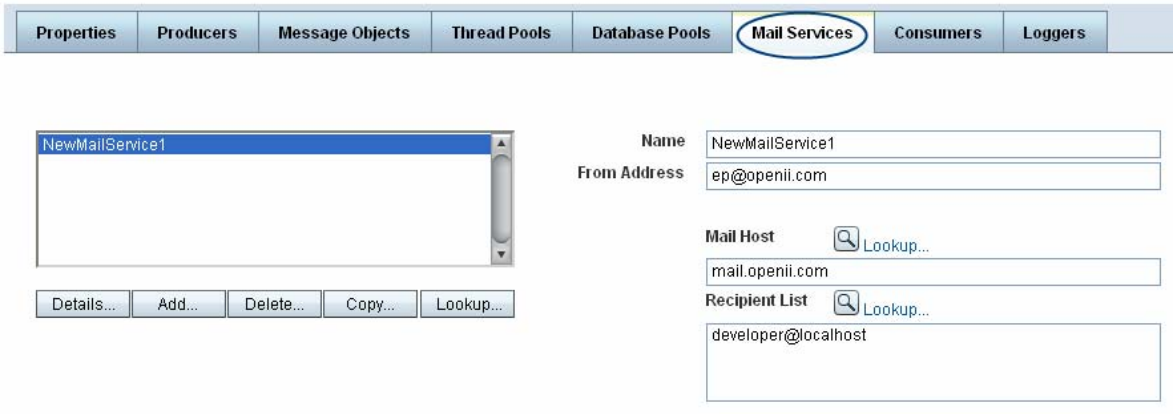
The following information may be specified for the Database Connection pool on the ***Database Pools->JDBC Settings*** sub-tab:



- ***Driver Name*:** The name of the JDBC implementation to use for the connections in this pool. For example: `oracle.jdbc.driver.OracleDriver`.
- ***Connect String*:** The name of the target database to which the connections in this pool should connect. For example: `jdbc:oracle:thin:@localhost:1521:DB_NAME`. This will be a provider-specific value.
- ***User Name*:** The name of the user used to connect to the database.
- ***Password*:** The password associated to the user.
- ***Lookup Driver Name*** by clicking on the "Lookup" link next to the Driver Name text box. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of Driver Name values used by other applications. The user may select a value that they would like to apply to this Database Pool and click on the Apply Changes button to apply the changes.
- ***Lookup Connect String*** by clicking on the "Lookup" link next to the Connect String text box. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of Connect String values used by other applications. The user may select a value that they would like to apply to this Database Pool and click on the Apply Changes button to apply the changes.

Mail Services

When the user clicks on the "Mail Services" tab a list of existing mail service objects associated to the selected command is shown. A mail service is an OpenEAI-based component used by an application to send e-mail messages using a simple SMTP protocol. This object is made available to the application at runtime and can be retrieved by name in the same way as other components. By configuring the mail service at application initialization the developer does not have to do any additional coding in order to use the mail service, rather simply retrieve it just as any other configurable OpenEAI object.



The user may perform the following actions on the list of mail service objects associated to the command:

- **Select a mail service object and view/modify its details** by clicking the "Details" button. The user can then make changes to the configuration of the object (i.e., change the mail host, etc.).
- **Add a new named mail service object** and then configure it appropriately.
- **Select a mail service object and delete it.** When you do this, it removes the mail service from the list of available mail services available to the command. If the command refers to the mail service that you delete, it may have errors at runtime so deleting mail services should be done in coordination with the development of the application/command.
- **Select a mail service object and copy it using a new name.** This is basically the same as creating a mail service object from scratch, except that by doing this, existing configuration information is copied over from the selected object.
- **Lookup mail services** by clicking on the "Lookup" button. When the user clicks on the "Lookup…" button, they will be taken to the `Lookup Component page.` The user can then select from the dropdown list of mail service used by other applications to either Copy or Edit.
- **Lookup Mail Host** by clicking on the "Lookup" link next to the Mail Host text box. When the user clicks on the "Lookup…" link, they will be taken to the `Lookup Property page.` The user can then select from the dropdown list of Mail Host values used by other applications. The user may select a value that they would like to apply to this Mail Service and click on the Apply Changes button to apply the changes.

- **Lookup Recipient List** by clicking on the "Lookup" link next to the Recipient List text box. When the user clicks on the "Lookup…" link, they will be taken to the `Lookup Property page`. The user can then select from the dropdown list of Recipient List values used by other applications. The user may select a value that they would like to apply to this Mail Service and click on the Apply Changes button to apply the changes. **[MJ - 10/25/07]**

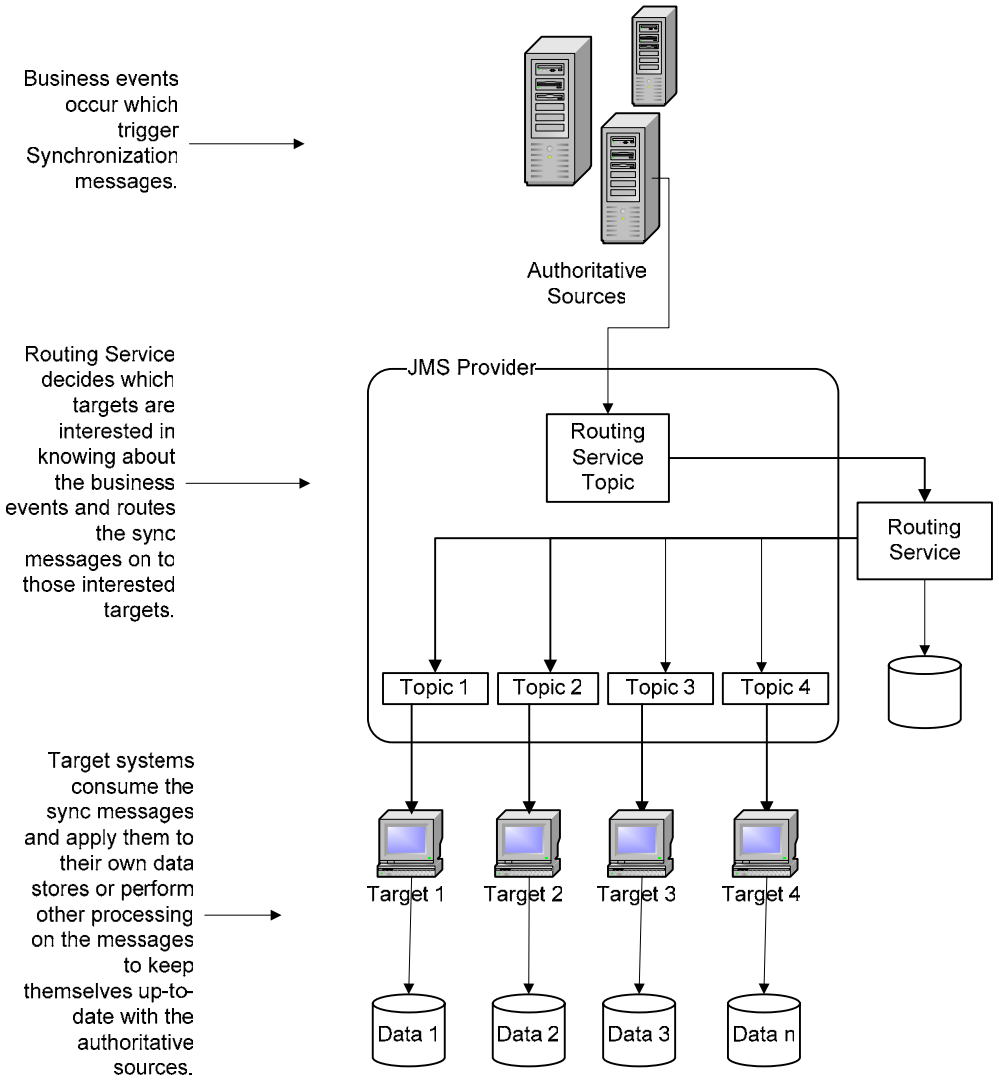
The following information may be specified for a mail service object:

- ***Name*:** The name of the Mail Service as it will be known by the `AppConfig` associated to the application or command. This is one of the ways that the developer of the application/command may choose to retrieve the object from `AppConfig`.
- ***From Address*:** The e-mail address of the sending application. This can be overridden at runtime if needed. This provides the administrator/developer the ability to specify the address here so the application does not have to supply it.
- ***Mail Host*:** The SMTP host that will be used to send the e-mail message.
- ***Recipient List*:** Comma separated list of recipient e-mail addresses to which a message will be sent.

Enhanced

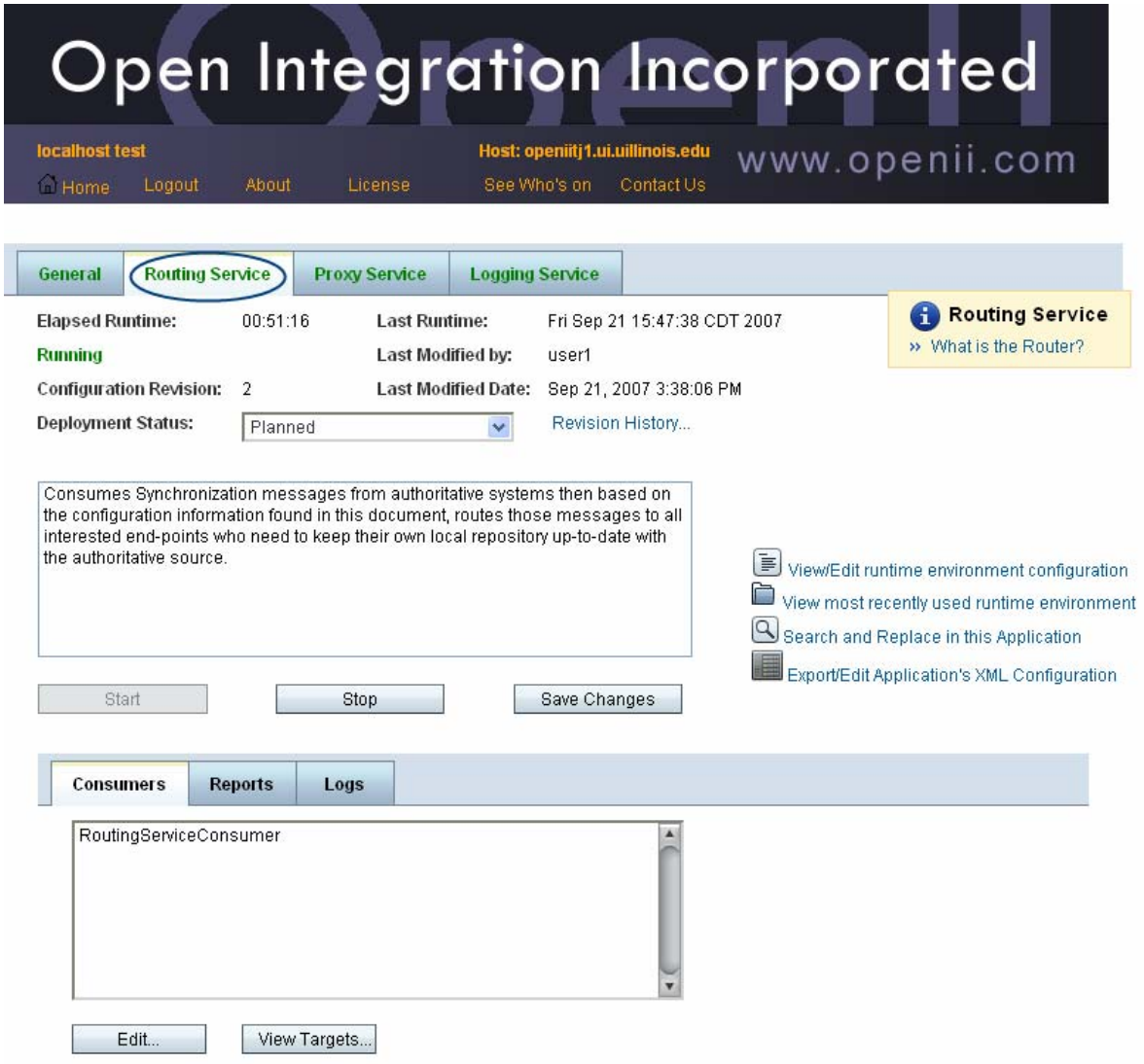
The Routing Service

As indicated in the Router Overview figure below, all sync publishing applications (i.e., all authoritative systems) send their messages to the Router's topic. Then, the router determines where those message needs to be delivered. It does this first by checking to see what targets, if any, are interested in the message object included in the message that is delivered to it. It looks at the message object and message action associated to the message for this first check. Additionally, more complex routing criteria can be specified. These can be very complex Java implementations, which may be target-specific. These components can look at the content of the message, data in other system or a variety of other criteria to determine if the target is really interested in the message. Future releases of the console will allow the user to maintain this information as well as the more "simple" criteria.



Enhanced

The Routing Service Page



This page will present the user with an interface to administer the Routing Service. The Router is an OpenEAI-based gateway that routes messages from one system to other targets, based on criteria appropriate for that target. The Router is "middleware" that centralizes this core EAI function. It is really just an OpenEAI-based gateway that serves a very specific purpose. Therefore, much of the Router configuration is similar to any general gateway. However, because of the specific purpose of the Router and the complexity associated with its configuration and maintenance, the Console provides some specific features related to Router configuration.

After the user logs into the application, they can select the "Routing Service" tab from the main navigation bar at the top of the page. This will take them to the `Routing Service Page`. On the

`Routing Service Page`, the user is presented with the high-level configuration information associated to the Router. The Router is configured via an entry in the t_app_config table in the database. This is the entry that is being manipulated via the Console. It is typically not necessary to modify this entry manually (i.e., outside of the console).

On the `Routing Service Page`, the user is presented with several pieces of information regarding the Router. These include:

- **The status of the Router** (**Running** or **Stopped**).
- **The description associated to the Router** which the user may change.
- **The amount of time the Router has been running (if it's running)**
- **The last time the router was running**
- **The last time the router was modified**
- **Last user associated with modfiying the router**
- **Configuration revision**

In addition, other "gateway specific" functions are available. The user has the ability to perform the following functions related to the Router on the `Router Page`:

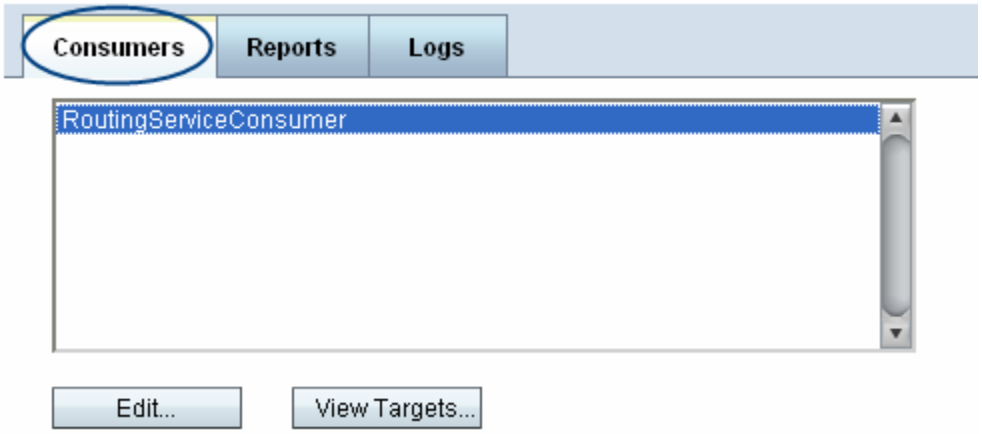
- **Start the Router by clicking on the "Start" button.** When you start the router via the console, it will initiate a new native process to run it in. Because of this, it is important to remember that when you start the router from within the console, you should always stop it using the console (when it needs to be stopped that is). The router is designed so that hopefully it should rarely need to be stopped but when it is necessary to stop it, remember to stop it via the console. This will prevent "orphan" processes that will remain running even when/if the console is closed or even if the application server is stopped. This behavior may vary based on the operating system being used but in general, any gateway/application that is started via the console should also be stopped via the console. If for some reason you forget to stop a gateway/application via the console, you can also look for the specific java process that was initiated and kill it manually but that is generally not recommended unless it's necessary. Refer to what happens when an application or gateway is started section of this document for more information.
- **Stop the Router by clicking on the "Stop" button.** This will terminate the process associated to the Router.
- **Save any changes made to the configuration of the Router** (for example, the description that can be changed on the Router's main page). Then, **when the Router is restarted**, it will use the new settings.
- **View the consumers associated to the Router.** Remember, router is just an OpenEAI based gateway that performs some very specific EAI "infrastructure" related functions. Therefore, at the most fundamental level, it behaves just like any other gateway (i.e. – it consists of consumers that execute commands when a message is delivered to the topic it's connected to).
- **View one of several canned reports** that provide information related to the processing that has been performed by the Router at a given point in time. **[TODO add more info and screen shots related to the Router reports]**
- **View and configure the log** associated to the router.
- **Deployment Status - ?**
- **Revision History**

  - The user is taken to a page that displays a table of the complete revision history for the selected application. This is useful for keeping track of who makes changes to the applications managed by this instance of the ESB.

- **View/Edit runtime environment configuration**
  - The JVM Process Properties page should open. Everything should be blank except for the Process Identifier and Console Identifier text fields. Those fields should be disabled.
- **View most recently used runtime environment**
  - The Recently Used Runtime Environment page should open. There should be six categories of runtime information displayed:
    - JVM Arguments
    - Application Id
    - Application Runner
    - Java Home
    - Classpath
    - Config element file spec
  - All items should be collapsed initially.
- **Search and Replace in this Application**
  - Opens the Search and Replace page. Use this interface to perform mass "search and replaces" in the configurations of applications being managed by this ESB.
  - IMPORTANT: This action should be performed with extreme caution
- **Export/Edit application's XML Configuration**
  - The Export/Edit Application Configuration page should open. The text area in the middle of the page should be populated with XML content that is the configuration for this application.
  - Warning: Changes made on this screen should be made with caution. Click the Edit button to manually edit this application's configuration. Click the Save button to save the changes made to the configuration.

- **Routing Service - Consumers Tab**

When the user selects the Consumers tab, a list of consumers associated to the Router is presented. A consumer used by the Router is the same as any other `PubSubConsumer` used by any other OpenEAI-based gateway, it connects to a JMS Topic and executes business logic when a synchronization message is delivered to that topic. This particular consumer executes the Router's command, which implements the business logic associated to the Router (namely, routing messages to one or more targets). Once the user has selected a consumer from the list, they may perform the following actions on that consumer:

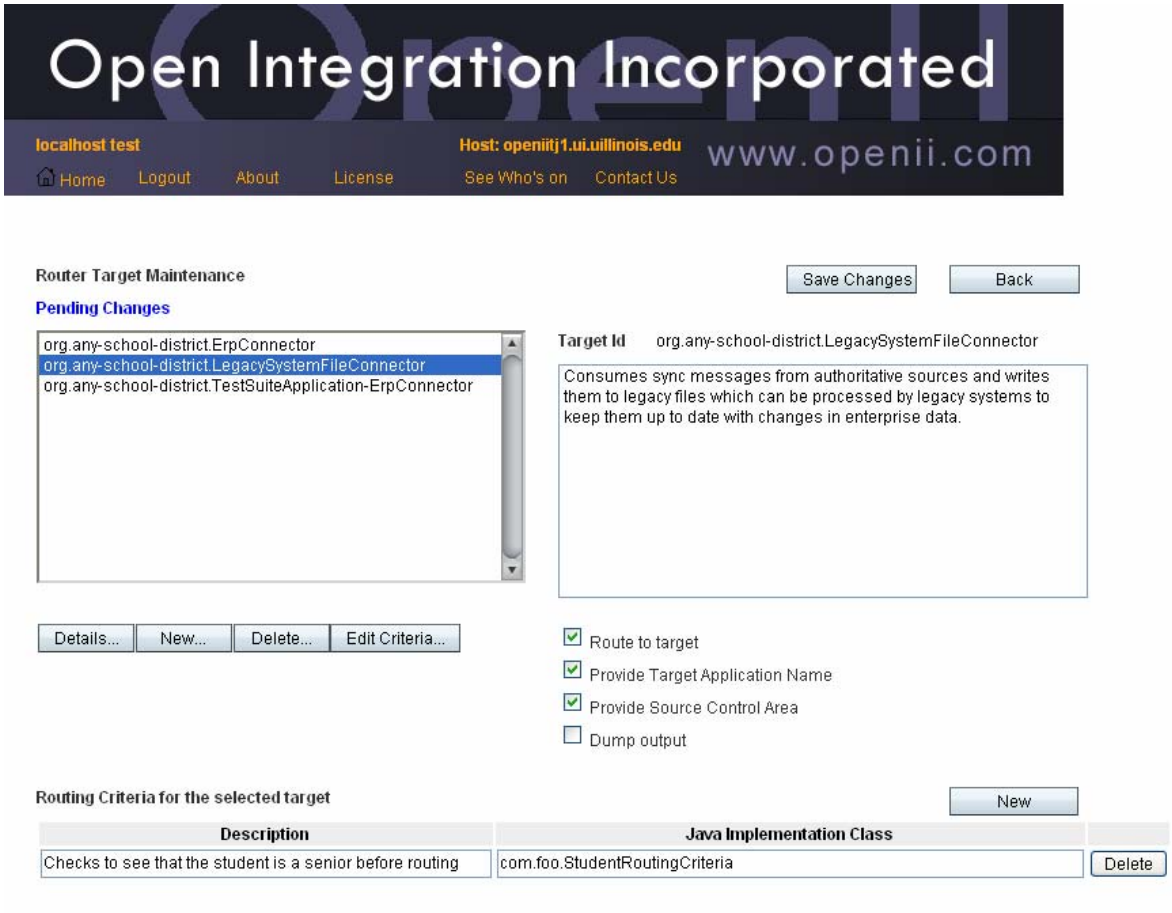


- **Edit the consumer's configuration** by clicking on the "Edit" button. When the user clicks on the "Edit" button, they will be taken to the `Consumer Details Page` (see the section titled "The Consumer Details Page") where they can change specific information related to the consumer itself. **[TODO – add more information about the detailed configuration of the Router's consumer]**
- **View the targets associated to the Router's consumer(s)** by selecting a consumer and clicking on the "View Targets" button. This will take the user to the `Router Targets Page` where the user will be presented with a list of targets (i.e. – end points) currently being routed to by the Routing Service. This is a list of other applications in the enterprise that are "interested" in messages being published by some authoritative source and being routed by the Router. See next section, titled "The Router Targets Page", for more information. When the Router consumes a message, it will consult this list to determine where to route the message consumed. Router does this by using other OpenEAI foundation components called `PubSubProducers` to re-publish the synchronization message to the appropriate targets.

Routing Service – Reports Tab

When the user selects the Reports tab…

The Router Targets Page



The `Router Targets Page` is where the user manages the targets being routed to by the selected consumer (the consumer that was selected on the Routing Service page). This page shows a list of all targets currently being routed to by the selected consumer. Remember, each consumer in a gateway executes commands, and the Router's command simply routes messages from authoritative systems to non-authoritative systems. Since organizations may find it necessary to have multiple consumers configured for the Router, they must first select the consumer they want to maintain.

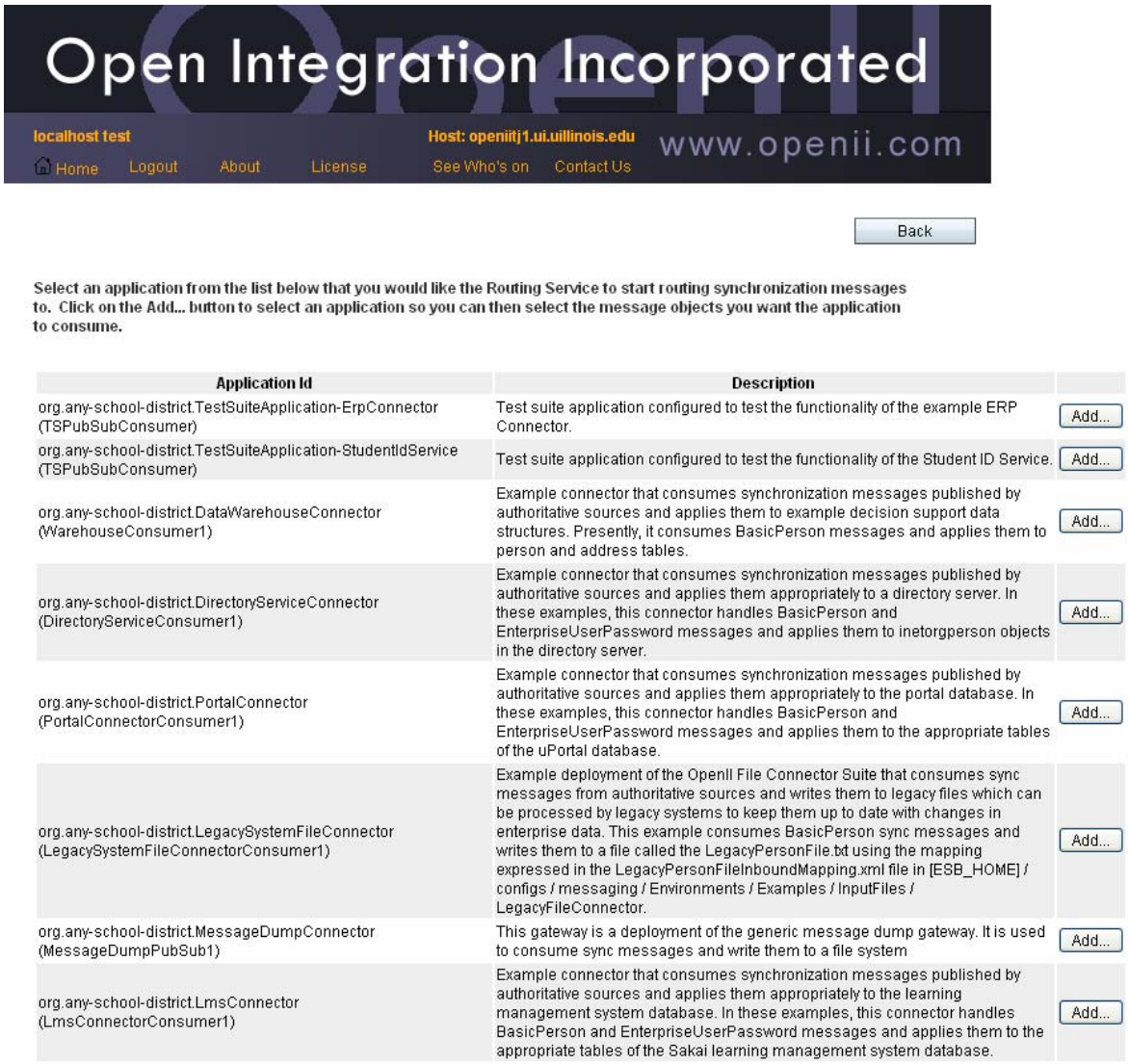
The first item of interest on this page is the list of existing targets being routed to. This is a list of existing sync consuming gateways that need to have certain types of messages delivered to them. The user may perform the following activities related to targets:

- **View general information about a target** by selecting a target and clicking the "Details…" button (or double clicking the desired target). Clicking the details button fills in the Target ID and Description fields appropriately. The user may modify the description of the target if they wish. Also, the general routing properties will be populated according to the current configuration associated to the target. These general routing properties include:
    - ***Route to Target*** **(true or false):** If "true" (checked) the Router will route to this target. If "false" (unchecked) the Router will not route any messages to the selected target but the target will still be available to the Router.
    - ***Provide Target App Name*** **(true or false):** If "true" (checked) Router will fill out the `TargetInfo` section of the `ControlArea` in the message it routes to the end-point (the target). This information is useful to other infrastructure applications like the Logging Service. It is used to correlate messages to their intended targets.
    - ***Provide Source Control Area*** **(true or false):** If "true" (checked) the Router will add the `ControlArea` element from the original message to the `SourceInfo` element in the `ControlArea` of the message being sent to the end-point (the target). This can be useful if the application needs to know things about the original publisher of this message (the place where the message originated).
    - ***Dump Output*** **(true or false):** If "true" (checked) the Router will write the content of the message it forwards to the file system. This is useful during development and testing to know exactly what message is being routed. By using this in combination with some of the other command options (such as the `writeToFile` attribute associated to a command) organizations can compare the message consumed by the Router to the message it forwards to the target. Router will use the MessageDumpDirectory associated to its command as the base for this directory. It will write these files to "[MessageDumpDirectory]/output".
- **Create a new target** by clicking on the "New…" button. This will take the user to the `Add Target Page` (see below) where the user can select from available sync consuming gateways that are not already being routed to and add them if appropriate. When these targets are added, Router is given the required `PubSubProducer` it needs to re-publish, or route the message to the target.
- **Delete the selected target** by clicking on the "Delete" button. The next time the Router is restarted, it will no longer route to that target.
- **Save any changes** made to the target ID, description, or general routing properties associated to the selected target(s) by clicking on the "Save" button.
- **Edit the routing criteria associated to the selected target** by clicking on the "Edit criteria for selected target" button. This will take the user to the `Select Routing Criteria Page` where the user can select the appropriate message object/action combinations that

the Router will use to determine if a particular message should be routed to the selected target. See the section titled "The Select Routing Criteria Page" for more information.

TODO: Routing criteria/rules that can be "plugged in"

The Add Router Target Page



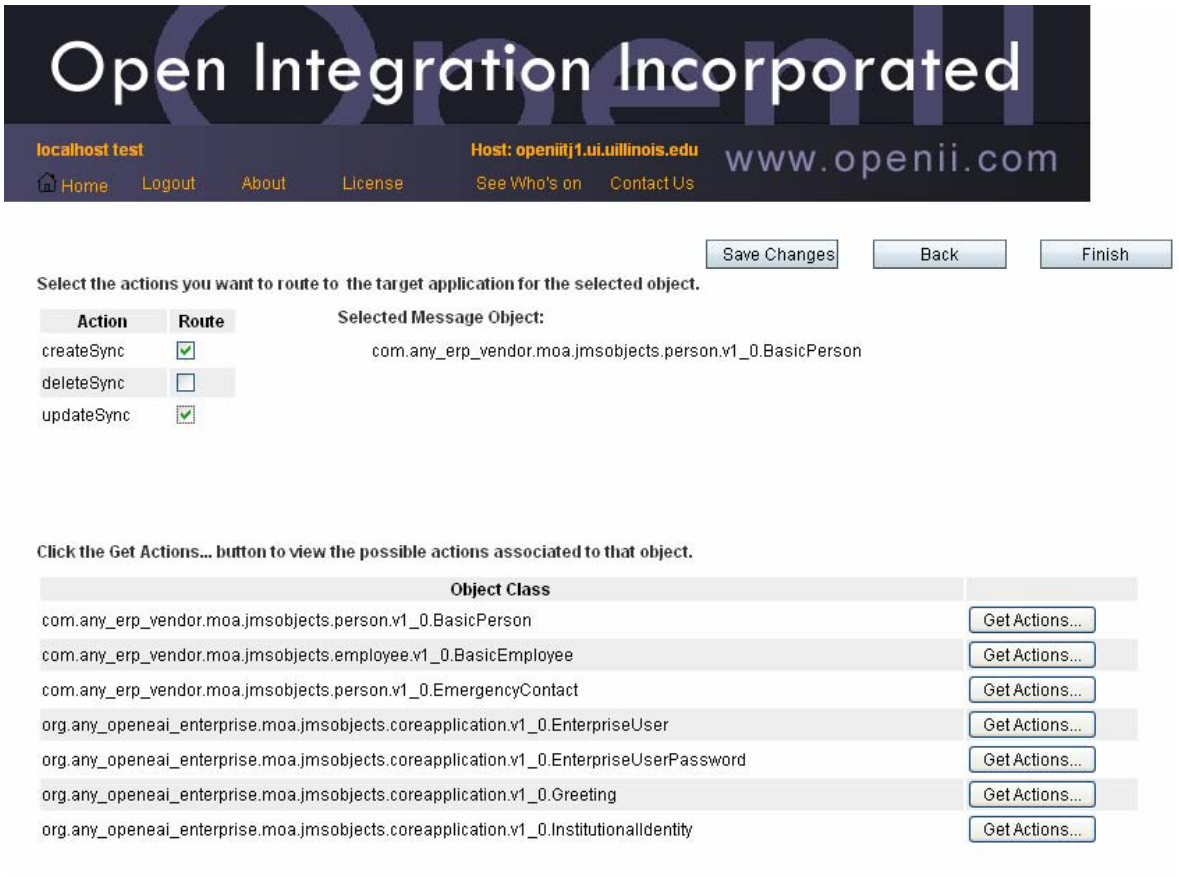
The `Add Router Target Page` allows the user to add a new target that should be routed to by the Router. When the user clicks on the "New…" button on the `Router Targets Page`, they will be taken here and presented with a list of potential targets. This list is built from existing configuration information associated to the gateways in an organization (i.e., an entry in the t_app_config table in the database that is manipulated via the Console)**.**

The Console examines the configuration associated to each sync consuming gateway and determines if that gateway is a suitable target. If so, it is listed here. The potential target's application ID and description is presented so the user knows for which application they are

looking. Once the desired application is found in the list, the user clicks the "Add Target…" to add that target. This will take the user to the `Select Routing Criteria Page` where the user can choose the message object/action combinations to be routed to the selected target.

TODO: mention that the list of apps being displayed may include multiple consumers so you have to choose the appropriate one that you really want router to route the message to.

The Select Routing Criteria Page



The `Select Routing Criteria Page` is where the user tells the Router which message objects it should route to the selected target (whether it is a new target or an existing target). This is the first set of criteria used by the Router when it determines if a message should be routed to a target or not. It looks at the incoming message object along with the action associated to it and uses that information to determine if a given target should receive that message. There are also other more complex routing criteria that may be specified that will be maintainable in future releases of the console.

The user is presented with a list of available objects in which the selected target is interested, based on the configuration of that target (i.e., the sync consuming gateway). The console looks at the configuration for that gateway and determines which actionable message objects are of interest. The console then builds the list of message objects. The user can then select the appropriate actions to route by clicking the "Get Actions…" button associated to the message object from the list. They can then select the appropriate actions that they want to allow through by checking or un-checking the checkboxes next to the list of actions associated to the selected message object.

Note: *at this time the target gateway must be configured with all the message objects in which it is interested. So, this means before a gateway can be added as a target, it must be first configured with the message objects it will need. The appropriate message object/action combinations can then be selected on this screen. In other words, if the message object is not already associated to the target gateway, it will not be listed as an object that can be selected on this page. However, if*

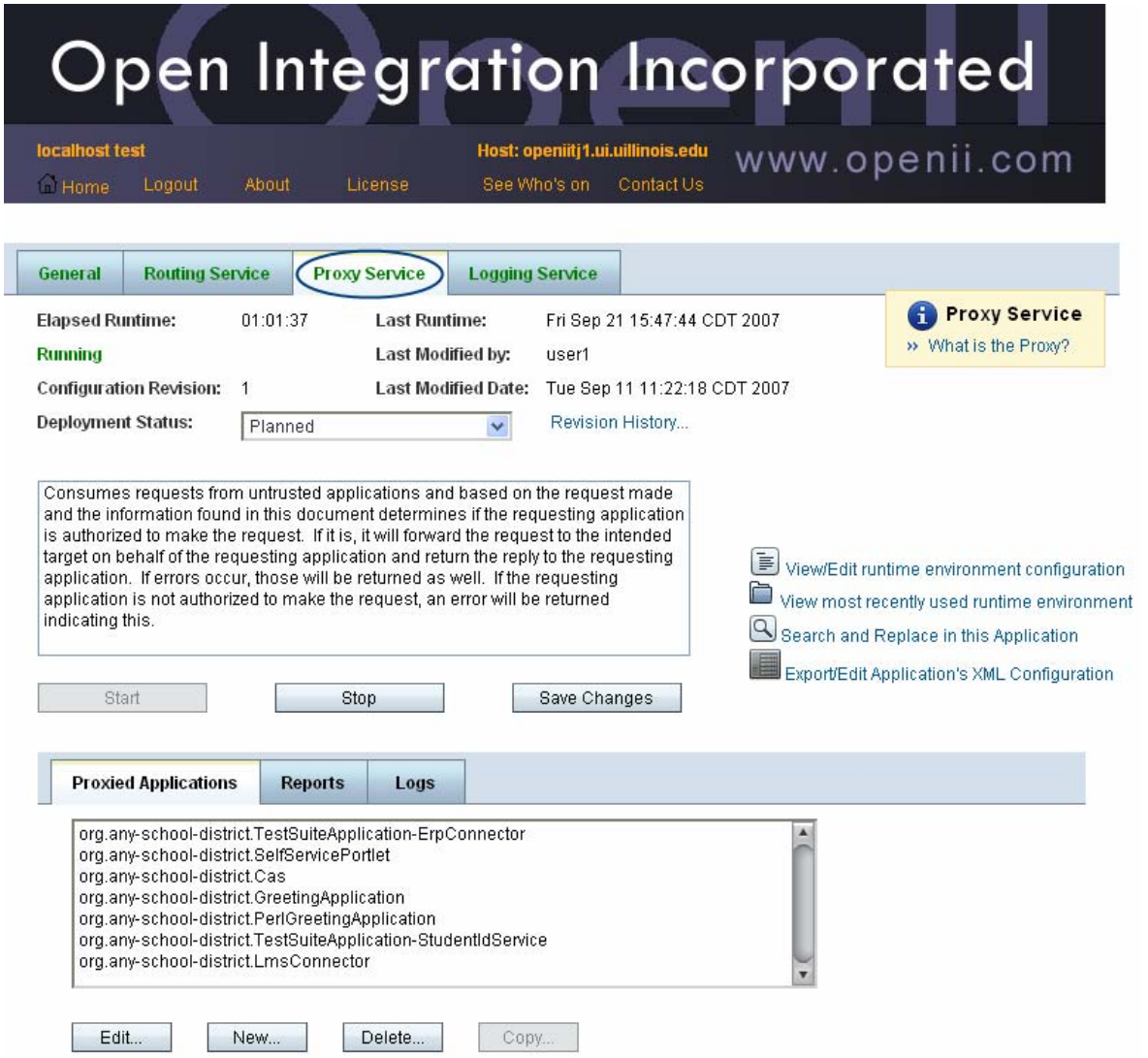
the consuming gateway does not currently have the selected message object/action combination in its configuration, the Console will automatically add it so there will be no need to update the target's configuration for this.

Once the appropriate routing criteria have been established, the user may click the "Save" button and the configuration document for the Router will be updated accordingly. The next time the Router is restarted, it will start routing to the selected target based on the rules specified here.

TODO: Finish Button

Enhanced

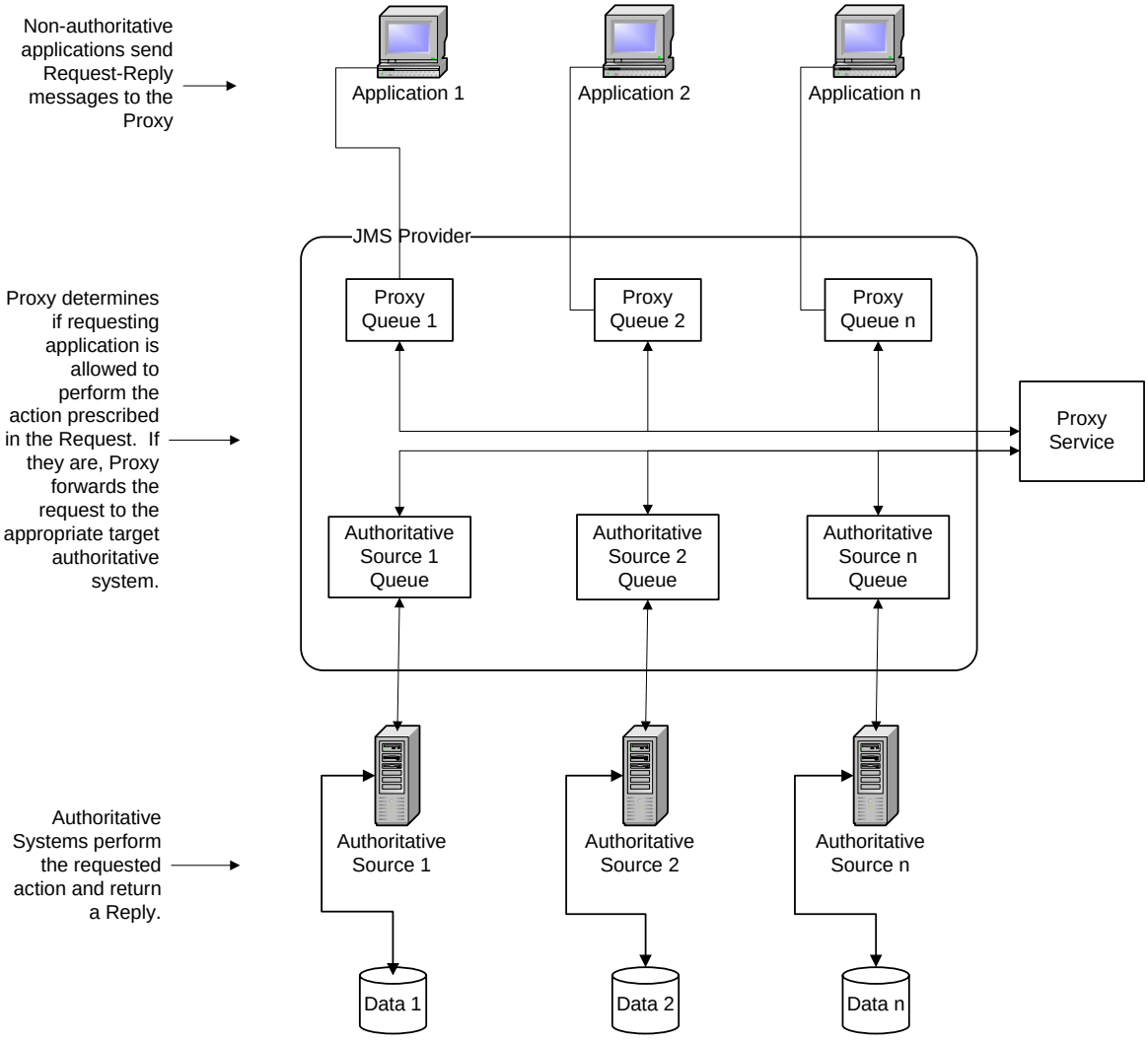
The Proxy Service Page



This page will present the user with an interface to administer the Proxy service. The Proxy is an OpenEAI-based gateway that sits between a requesting application and an authoritative source. The Proxy determines if the requesting application is permitted to perform the action it is requesting based on several criteria. The Proxy is a component of EAI middleware that centralizes this core function.

The Proxy is simply an OpenEAI-based gateway that serves a very specific purpose. Therefore, much of the Proxy configuration is very similar to that of any other gateway. However, because of the specific purpose served by the Proxy and the complexity associated to its configuration and maintenance, the Console provides some specific features related to Proxy configuration.

As the Proxy overview diagram below indicates, the purpose of the Proxy is to ensure that a requesting application is allowed to perform a given action on an object. How it determines this can be very simple or very complex. With this release, the console allows the user to choose which actions may be performed on a given object by a given "proxied" application via the interface. This is the first level of checking that occurs within the Proxy service. However, additional lower level rules can also be specified and developed that are "plugged" into the Proxy. When those lower level rules exist, the proxy not only checks to see that the requesting application has the authority to perform the action it's requesting on the object, but it will invoke these lower level rules as well to make the check potentially much more complex. In future releases of the console, the user will be able to associate these lower-level, more complex rules to the requesting application as well.



As indicated in the above diagram, there are JMS Queues established for each requesting application (a.k.a., proxy queues). These queues are the only queues the requesting application may send requests to. The Proxy consumes the request off of the queue and performs the check(s) to see if the sending application has the authority to perform the action on the selected object. If the application is allowed to perform the action, the request is forwarded on to the intended target (the Proxy Target application). Then, the Proxy consumes the Reply from the target application and returns it to the requesting application.

After the user logs into the application, the user can select the "Proxy Page" tab from the main navigation bar across the top of the page. This will take them to the `Proxy Page`. On this page, the user is presented with the high-level configuration information associated to the Proxy. The Proxy is configured via an entry in the t_app_config table in the database. It is typically not necessary to modify this entry manually (i.e. outside of the console).

On the `Proxy Page`, the user is presented with several pieces of information. These include:

- **The status of the Proxy** (**Running** or **Stopped**).
- **The description associated with the Proxy,** which the user may change.
- **The amount of time the Proxy has been running (if it's running)**
- **The last time the Proxy was run**
- **The last time the Proxy was modified**
- **Last user associated with modfiying the Proxy**
- **Configuration revision**

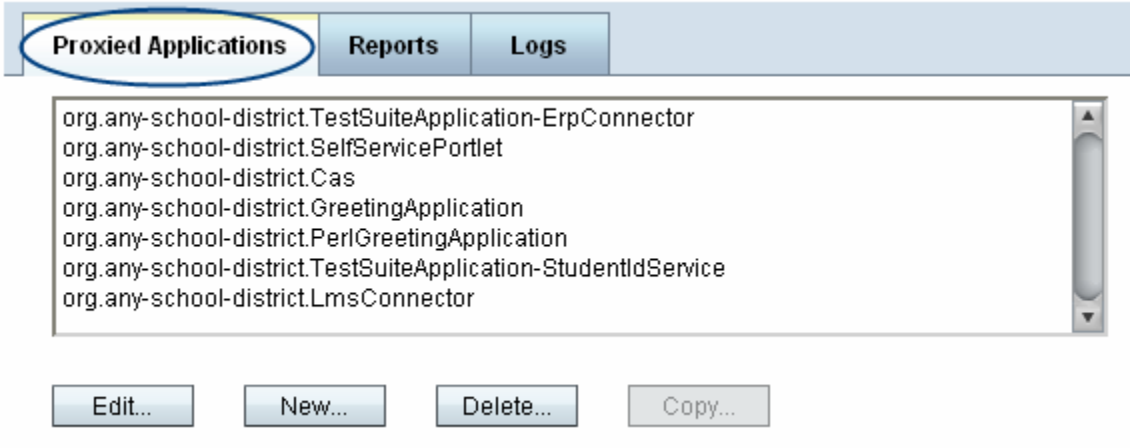
In addition to this information, other gateway-specific functions are available. The user has the ability to perform the following functions related to the Proxy on the `Proxy Page`:

- **Start the Proxy** by clicking on the "Start" button. When you start the proxy via the console, it will initiate a new native process to run it in. Because of this, it is important to remember that when you start the proxy from within the console, you should always stop it using the console (when it needs to be stopped that is). The proxy is designed so that it should rarely need to be stopped but when it is necessary to stop it, remember to stop it via the console. This will prevent "orphan" processes that will remain running even when/if the console is closed or even if the application server is stopped. This behavior may vary based on the operating system being used but in general, any gateway/application that is started via the console should also be stopped via the console. If for some reason you forget to stop a gateway/application via the console, you can also look for the specific java process that was initiated and kill it manually but that is generally not recommended unless it's necessary. Refer to the what happens when an application or gateway is started section of this document for more information.
- **Stop the Proxy** by clicking on the "Stop" button. This will terminate the process associated to the Proxy.
- **Save any changes** made to the configuration of the Proxy (the description for example that may be changed on the Proxy's main page). When the Proxy is restarted, it will use the new settings.
- **View the existing proxied applications associated to the Proxy** by clicking on the "Proxied Applications" tab. When the user clicks this button the "Applications currently proxied by the Proxy" form is displayed, which lists all request sending applications that are currently configured to send requests through the Proxy on their way to some other end point. See the following section titled "View Proxied Apps" for more information.
- **View and configure the log file associated to the proxy**.
- **Deployment Status - ?**
- **Revision History**
    - The user is taken to a page that displays a table of the complete revision history for the selected application. This is useful for keeping track of who makes changes to the applications managed by this instance of the ESB.

- **View/Edit runtime environment configuration**
  - The JVM Process Properties page should open. Everything should be blank except for the Process Identifier and Console Identifier text fields. Those fields should be disabled.
- **View most recently used runtime environment**
  - The Recently Used Runtime Environment page should open. There should be six categories of runtime information displayed:
    - JVM Arguments
    - Application Id
    - Application Runner
    - Java Home
    - Classpath
    - Config element file spec
  - All items should be collapsed initially.
- **Search and Replace in this application**
  - Opens the Search and Replace page. Use this interface to perform mass "search and replaces" in the configurations of applications being managed by this ESB.
  - IMPORTANT: This action should be performed with extreme caution
- **Export/Edit Application's XML Configuration**
  - The Export/Edit Application Configuration page should open. The text area in the middle of the page should be populated with XML content that is the configuration for this application.
  - Warning: Changes made on this screen should be made with caution. Click the Edit button to manually edit this application's configuration. Click the Save button to save the changes made to the configuration.

View Proxied Apps

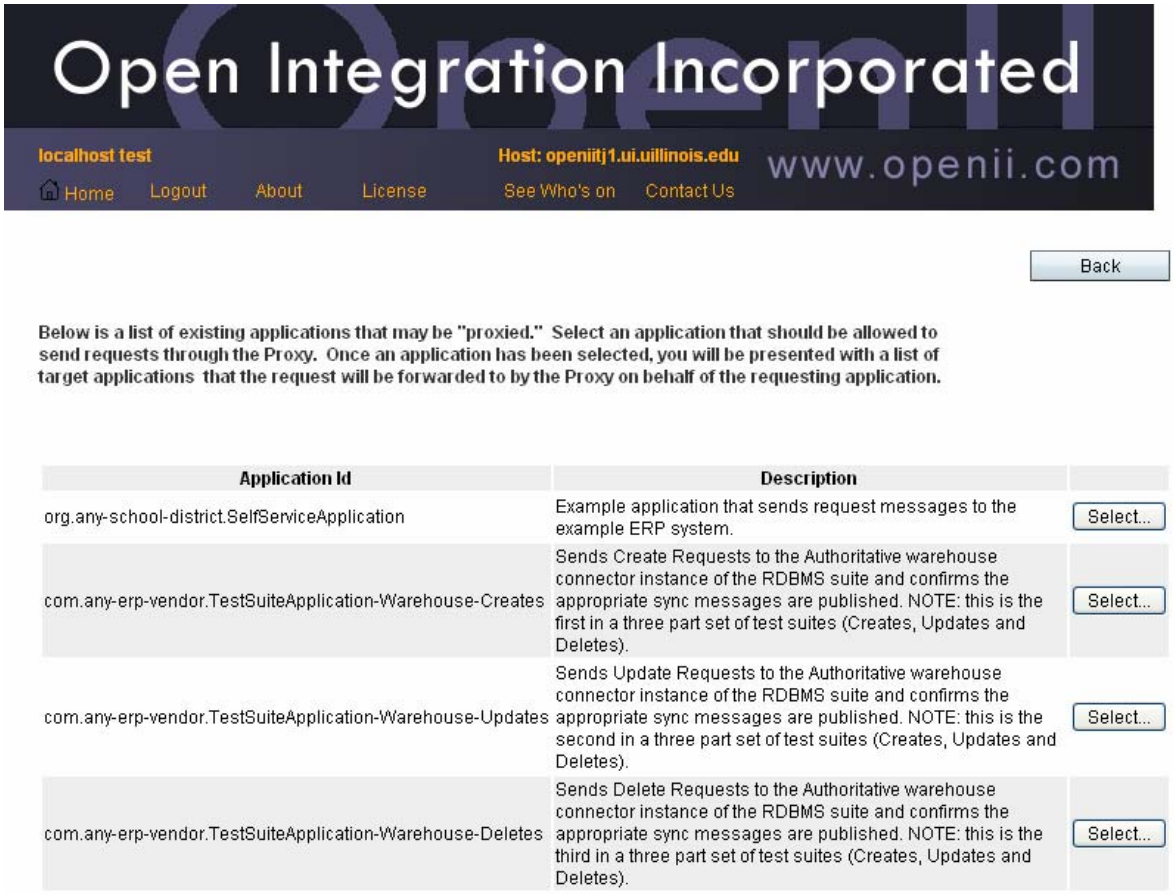
When the user clicks the "Proxied Applications" tab, they will be presented with a list of applications that are currently configured to send requests through the Proxy on their way to some other end point. This means, the Proxy is being used to make sure the requesting application is sending only "allowed" messages to the end point. The Console provides interfaces to aid in the administration of the rules used to make this determination; much like the Routing Service does for PubSub messages.



In the Proxied Applications tab, the user may perform several functions:

- **Administer the proxy rules associated to a requesting application** by selecting a proxied application from the list and clicking on the "Edit…" button. This will take them to the `Proxied Application Maintenance Page`, where the user can view and modify information related to this particular proxied application, for instance, which consumers are used by the Proxy to proxy the requests coming from the requesting application, as well as to which targets the requesting application is allowed to send requests. See the section titled "Proxied Application Maintenance Page" for more information.
- **Create a brand new proxied application** by clicking on the "New…" button. This will take the user to the `Add Proxied Application Page`, where the user can select a potential proxiable application in advance of specifying the proxy rules associated to the requesting application that must be in place. See the section titled "Add Proxied Application Page" for more information about this process.
- **Delete the selected proxied application** by clicking the "Delete" button. This will remove that application and any consumers used by the Proxy for that application. When this is done, the selected application will no longer be able to send ANY requests through the proxy so this is a convenient way to "shut off" a particular application.

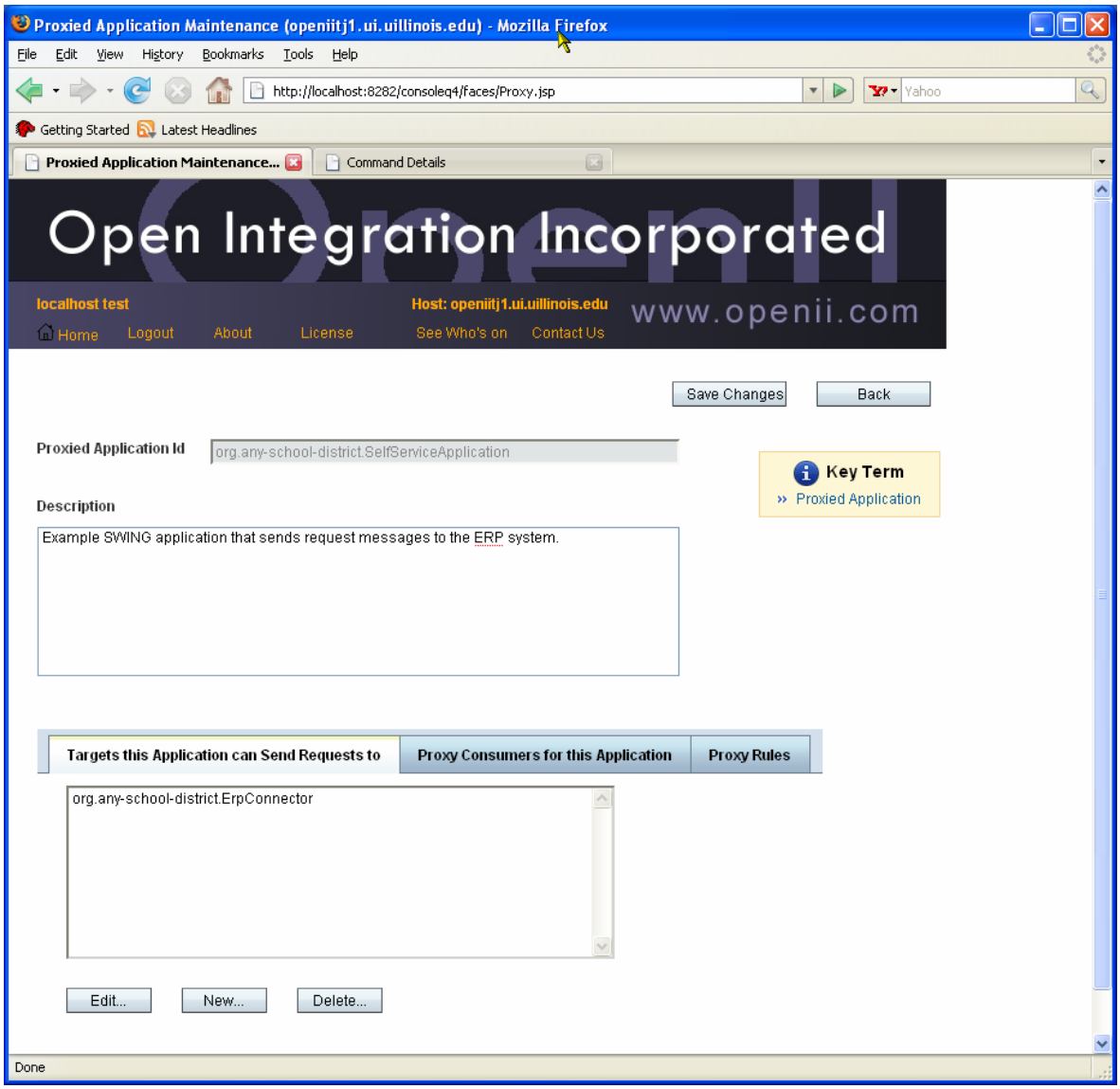
Add Proxied Application Page



The `Add Proxied Application Page` provides an interface to add a brand new requesting application to the Proxy (i.e., a new application that will be sending requests through the Proxy to some other end point). On this page, the user is presented with a list of applications that may be proxied. The Console builds this list based on existing information in the organization's general configuration document (the Deployment Descriptor). These are applications that have PointToPointProducers in their configuration and are performing at least one action on at least one actionable message object.

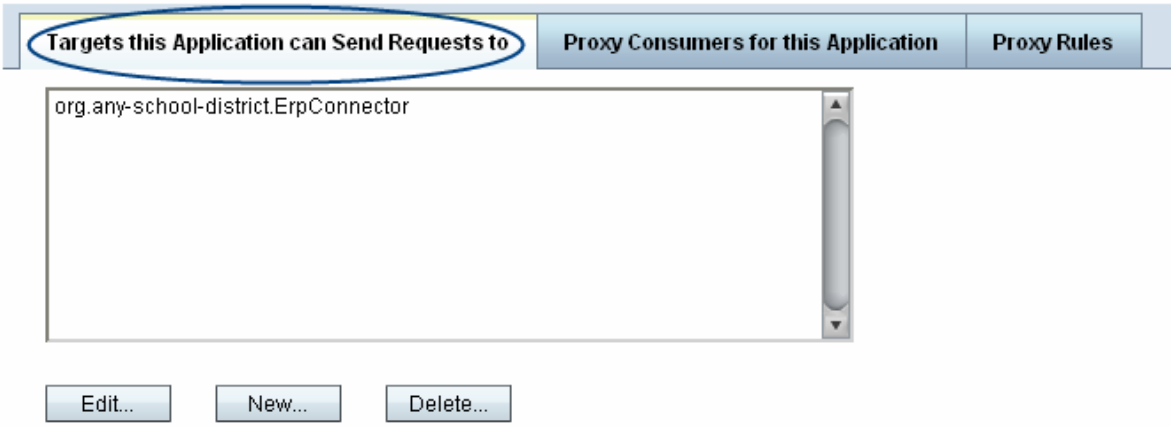
The application ID and description of the potential requesting application is displayed to the user. When the user clicks on the "Add…" button for an application, the user is taken to the `Proxied Application Maintenance Page` to specify targets and proxy criteria/rules that will be checked before allowing the application to send requests to the selected targets.

Proxied Application Maintenance Page

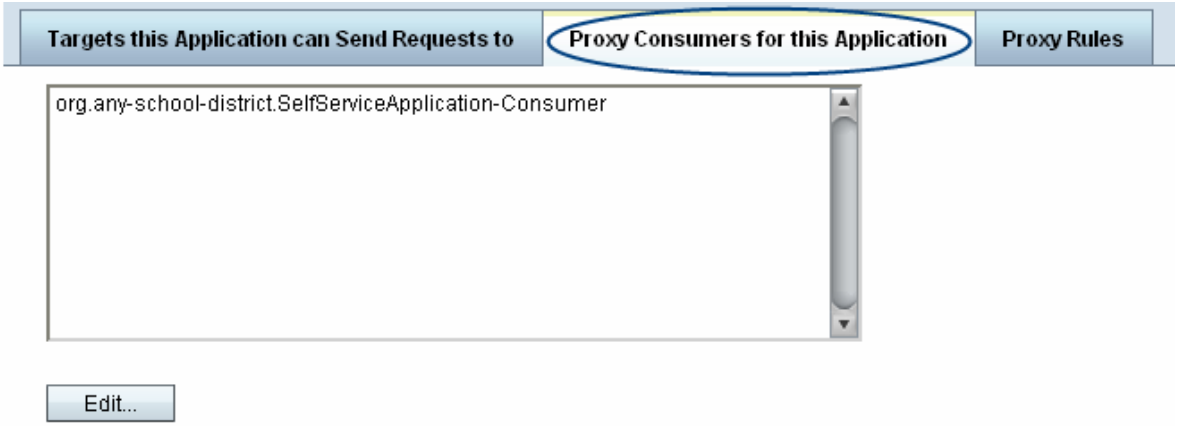


The `Proxied Application Maintenance Page` provides an interface for managing Proxy information related to the selected application (i.e., control what targets the selected application may send requests to). This page can be reached from either the `Proxy Page` when the user selects an existing proxied application and clicks the "Edit…" button, or during the process of adding a new proxied application (see `Add Proxied Application Page`). This page presents the user information about the proxied application, including the application ID and description of the selected proxied application.

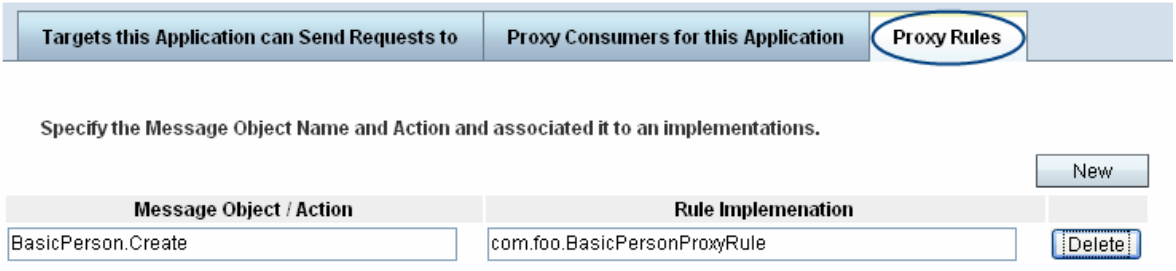
The user can perform several actions on this page related to the selected proxied app:



- **View/modify information about the targets** to which the selected application is allowed to send requests (see "Targets this Application can Send Requests to" below).



- **View/modify information about the consumers** used by the Proxy to proxy requests coming from the proxied application. Note: most of the information about these consumers is automatically built by the Console when a new application is added to the Proxy, so it is not generally necessary to modify much about these consumers except the Destination name that they are consuming from.



- **View/modify Proxy "rules"** that will be associated to this proxied application… **TODO: Proxy Rules Tab**

- **Change the application ID and description associated to a proxied application** and save it. Note: this information is also automatically derived by the Console when the application is added to the Proxy. So, it is not generally necessary to change much about these except to provide a more specific description that might indicate more about the nature of the proxy rules for the selected application.

Targets this Application can Send Requests to

When the user clicks on the "Targets this Application can Send Requests to" tab, the targets to which the selected application is allowed to send requests are listed.

On this tab, the user can perform the following actions:

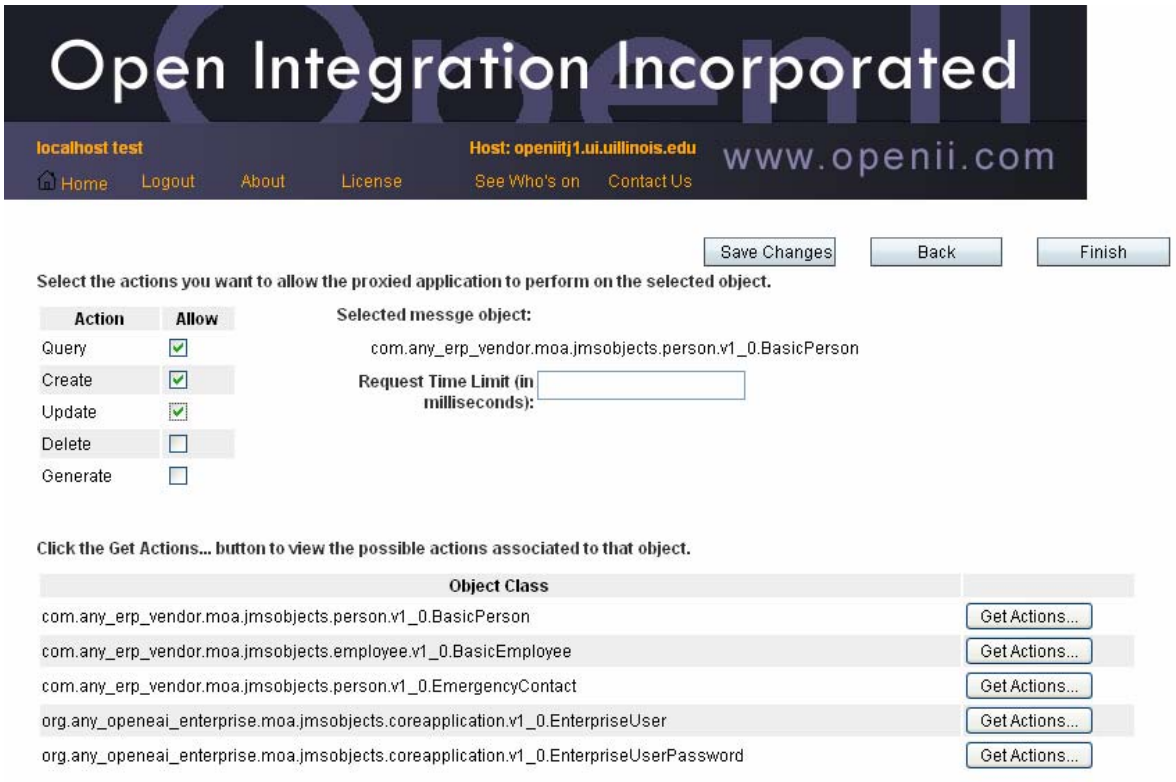
- **Edit the proxy rules associated to a given target** for the selected app by clicking on the "Edit…" button. This will take the user to the `Select Target Proxy Criteria Page`, which allows the user to select message object/action combinations that the requesting application will be allowed to send to the selected target. See the section titled "Select Target Proxy Criteria Page" for more information.
- **Add a brand new target** that the requesting application will be able to send requests to by clicking on the "New…" button. This will take the user to the `Add New Proxy Target Page`, where they may select a target from the list of available targets before specifying proxy rules (i.e., criteria).
- **Delete the selected target** by clicking on the "Delete" button. This will mean that the requesting application will no longer be able to send ANY requests to the selected target.

Add New Proxy Target Page

When the user chooses to add a new target to the selected proxied application, they are taken to the `Add New Proxy Target Page`. On this page the user can choose from a list of potential targets that are authoritative for some data (i.e., they have PointToPointConsumers and can process at least one actionable object). From this list, the user can select the target to which they wish to allow the requesting app to send requests. Subsequently, they will be taken to the `Select Target Proxy Criteria Page`, where they can specify the proxy criteria/rules that will be checked by the Proxy before allowing the requesting application to send a given request to the selected target.

This list is broken down by consumer, so if an application has multiple consumers, the user will need to be careful to pick the correct one.

Select Target Proxy Criteria Page



The `Select Target Proxy Criteria Page` is where the user tells the Proxy which message objects it should allow the requesting application to send to the selected target (whether it is a new target or an existing target). This is the first set of criteria used by the Proxy when it determines if a request should be allowed from a requesting application. It looks at the incoming message object along with the action associated to it, and uses that information to determine if the requesting application is allowed to send that particular request. Other more complex proxy criteria/rules could be specified as well; these will be maintainable in coming releases of the Console.

The user is presented with a list of available objects for which the selected target is authoritative based on the configuration of that target (i.e., the request-consuming gateway). The Console looks at the configuration for that gateway and determines for which actionable message objects it is authoritative. It then builds the list of message objects. The user can select the appropriate actions to allow by clicking the "Get Actions…" button associated to the message object from the list. The user can subsequently select the appropriate actions to be allowed through by checking or un-checking the checkboxes next to the list of actions associated to the selected message object.

Note: at this time the target gateway must be configured with all the message objects for which it is authoritative. Before a gateway can be added as a target, it must be configured with the required message objects. The appropriate message object/action combinations can be selected on this screen. In other words, if the message object is not already associated to the target gateway, it will not be listed as an object that can be selected on this page. However, if the requesting application does not currently have the selected message object/action combination in its configuration, the Console will automatically add it so there will be no need to update the requesting application's configuration for this.

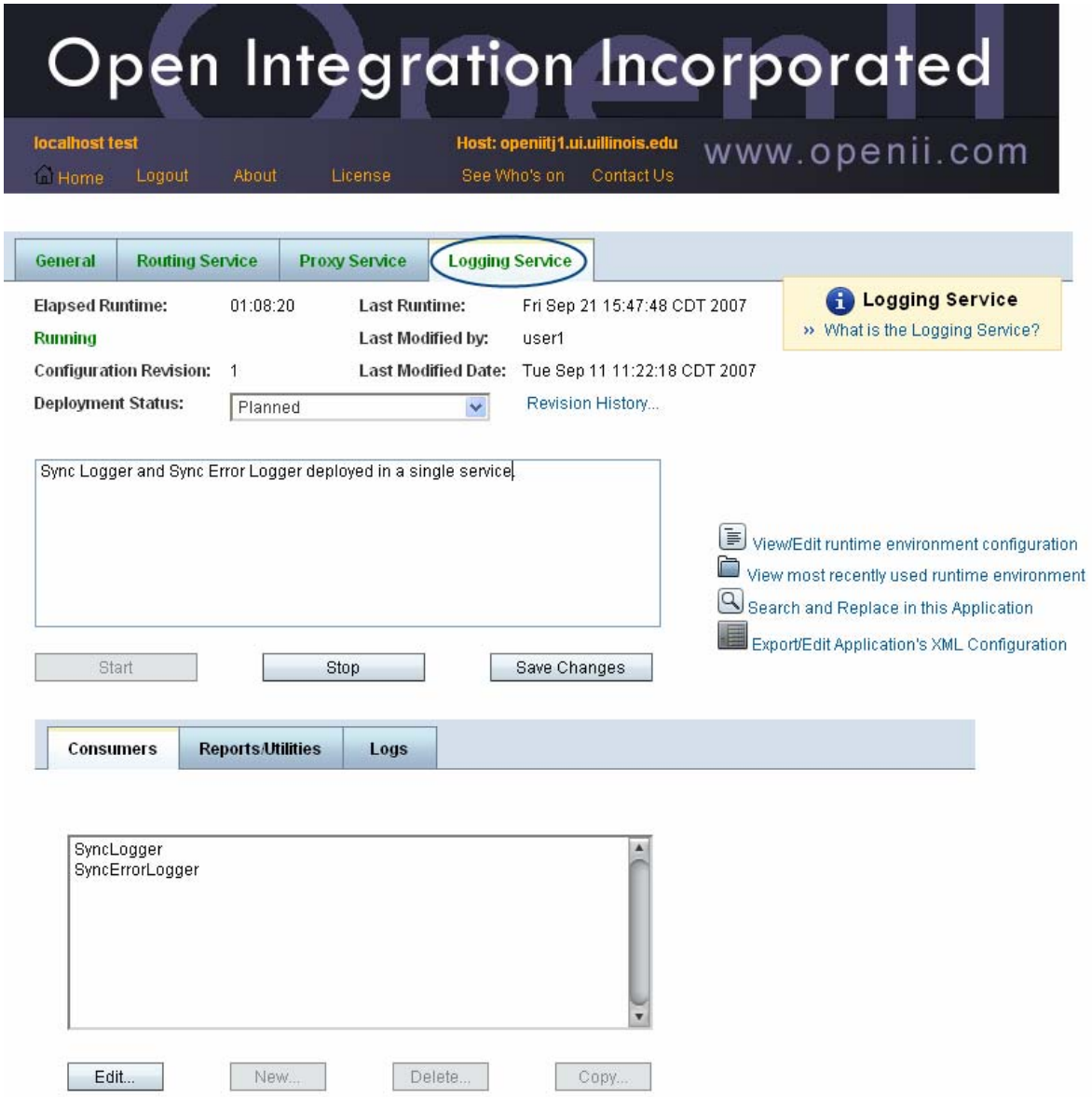
Once the appropriate proxy criteria/rules have been established, the user may click the “Save” button and the configuration document for the Proxy will be updated accordingly. The next time the Proxy is restarted, it will allow the requesting application to send requests to the selected target based on the rules specified here.

TODO: New Feature

- **Request Time Limit**
- **Finish button**

Enhanced

The Logging Service Page



The `Logging Service Page` will present the user with an interface to administer the enterprise logging service (a.k.a., ELS). The Logging Service is an OpenEAI-based gateway that consumes and processes all synchronization (sync) messages published by all sync-producing applications or gateways. Additionally, the Logging Service consumes Sync-Error-Sync messages published by sync-consuming gateways when they encounter an error processing a consumed message. So, like the Router and Proxy, the Logging Service provides another piece of EAI middleware. Organizations can view the messages that have been published by all applications and generate reports related to the messages being produced, and can view any errors and correlate those errors with the messages that caused them.

When an OpenEAI based sync-producing application publishes a message, it actually publishes that messages to two topics. First, it publishes the message to the Router topic as mentioned in the Router overview section of this document. Second, it publishes that same exact message to the ELS topic. It does this automatically without any additional coding needed. This is just part of the OpenEAI `PubSubProducer` foundation.

Likewise, when a sync-consuming gateway has errors processing a consumed message, it is required to publish a Sync-Error-Sync message. These messages are consumed by the `SyncErrorLogger` portion of ELS and persisted for later use.

The Logging Service is an OpenEAI-based gateway that serves a very specific purpose. Therefore, much of its configuration is very similar to any gateway. However, because of the specialized purpose served by the Logging Service, the Console provides specific features related to Logging Service configuration and administration.

Future releases of the console will provide various reports that can be used to keep track of the messages that are being published and additionally correlate error messages that are published to the message(s) that were being processed when the error occurred. Once the analysis has been performed and the cause of the problem remedied, the user will be able to republish those messages to the intended target.

[TODO - insert Logging Service overview diagram of some sort here, like the Router and Proxy overview diagrams]

After the user logs into the application, the user can select the "Logging Service Page" tab from the main navigation bar at the top of the page. This will take them to the `Logging Service Page`. On this page, the user is presented with high-level configuration information associated to the Logging Service. The Logging Service is configured via an entry in the t_app_config table in the database. This is the entry that is being manipulated via the Console. It is typically not necessary to modify this entry manually (i.e. outside of the console).

On the `Logging Service Page`, the user is presented with several pieces of information regarding the service. These include:

- **The "ID" associated to the logging service,** which the user may change (*future release*).
- **The status of the ELS** (**Running** or **Stopped**).
- **The description associated with the logging service,** which the user may change.
- **The amount of time the ELS has been running (if it's running)**
- **The last time the ELS was run**
- **The last time the ELS was modified**
- **Last user associated with modfiying the ELS**
- **Configuration revision**

In addition to these pieces of information, other "gateway-specific" functions are available. The user has the ability to perform the following functions related to the logging service on the `Logging Service Page`:

- **Start the ELS** by clicking on the "Start" button. When you start ELS via the console, it will initiate a new native process to run it in. Because of this, it is important to remember that when you start ELS from within the console, you should always stop it using the console

(when it needs to be stopped that is). ELS is designed so that it should rarely need to be stopped but when it is necessary to stop it, remember to stop it via the console. This will prevent "orphan" processes that will remain running even when/if the console is closed or even if the application server is stopped. This behavior may vary based on the operating system being used but in general, any gateway/application that is started via the console should also be stopped via the console. If for some reason you forget to stop a gateway/application via the console, you can also look for the specific java process that was initiated and kill it manually but that is generally not recommended unless it's necessary. Refer to the what happens when an application or gateway is started section of this document for more information.

- **Stop the ELS** by clicking on the "Stop" button. This will terminate the process associated to the Logging Service.
- **Save any changes** made to the configuration of the ELS (the description for example that may be changed on the ELS main page). When the ELS is restarted, it will use the new settings.
- **View the consumers** associated to the ELS. For ELS, there are generally at least two consumers. One consumer that simply persists all sync messages published and one consumer that persists the Sync-Error-Sync messages mentioned previously.
- **View one of several canned reports** that provide information related to the processing that has been performed by the ELS at a given point in time **[TODO: more on this]**.
- **View and configure the file associated to the Logging Service**.
- **Deployment Status - ?**
- **Revision History**
  - The user is taken to a page that displays a table of the complete revision history for the selected application. This is useful for keeping track of who makes changes to the applications managed by this instance of the ESB.
- **View/Edit runtime environment configuration**
  - The JVM Process Properties page should open. Everything should be blank except for the Process Identifier and Console Identifier text fields. Those fields should be disabled.
- **View most recently used runtime environment**
  - The Recently Used Runtime Environment page should open. There should be six categories of runtime information displayed:
    - JVM Arguments
    - Application Id
    - Application Runner
    - Java Home
    - Classpath
    - Config element file spec
  - All items should be collapsed initially.
- **Search and Replace in this application**
  - Opens the Search and Replace page. Use this interface to perform mass "search and replaces" in the configurations of applications being managed by this ESB.
  - IMPORTANT: This action should be performed with extreme caution
- **Export/Edit Application's XML Configuration**

- The Export/Edit Application Configuration page should open. The text area in the middle of the page should be populated with XML content that is the configuration for this application.
- Warning: Changes made on this screen should be made with caution. Click the Edit button to manually edit this application's configuration. Click the Save button to save the changes made to the configuration.

ELS Consumers Tab

When the user selects "Consumers" tab, a list of consumers associated to ELS is presented. As mentioned above, there are generally two consumers associated to the ELS (one that persists all sync messages published and one that persists Sync-Error-Sync messages that are published when sync consuming gateways encounter errors processing a message they have consumed. The consumers used by the ELS are just like any other `PubSubConsumer` used by any other OpenEAI-based gateway. This particular consumer executes the specific ELS commands that implement the business logic associated to the ELS (namely, logging the sync messages mentioned previously). Once the user has selected a consumer from the list, they may perform the following actions on that consumer:

- **Edit the consumer's configuration** by clicking on the "Edit" button. When the user clicks on the "Edit" button, they will be taken to the `Consumer Details Page`, where they can change specific information related to the consumer itself. **[TODO: more information on the detailed consumer/command configuration for logging and error logging services]**

TODO: More screen shots and documentation regarding Logging Service reports/utilities

What happens when an application or gateway is started?

When an application or gateway is started via the Console a new process is spawned to run that component. This is just an invocation of a JVM (i.e., a Java execution). When this process is started, the console, dynamically builds a `CLASSPATH` that will be used to run the process. The `CLASSPATH` will include all libraries that needed to start the gateway or application. In order to build this `CLASSPATH`, the console needs to know where the libraries are associated to that application (i.e., .jar or .zip files that contain the required Java classes). In the Console `web.xml` file, there is another `context-param` that is used to tell the console this location. This parameter is called `defaultRuntimeLibPaths` and should contain a colon or semi-colon delimited list of paths containing runtime libraries (jars and zip files) that are required by applications that may be deployed into and managed by the console. The console will parse this list of paths and build the `CLASSPATH` appropriately based on what it finds in those paths.

Because the console builds this `CLASSPATH` dynamically when you start a component, it will automatically detect new or changed applications that are added to any of these `additionalRuntimeLibPaths` without a bounce or re-start of the application server.

This process is a standard, platform independent `java.lang.Process` object. If the process successfully starts, the console maintains a reference to this process (which is how it stops it later if needed). Because this is a spawned process, it will not go away when the user's browser is closed or even when the application server is stopped. Therefore, it is important to remember that when you start an application/gateway via the Console, you should stop it via the console. If for some reason that doesn't happen, the process will keep running and can only be stopped by finding the process in the process table and forcibly terminating it.

[TODO – Redo this section to take into account the dynamic application configuration features of the console. T_app_configs and t_app_settings]

Reports/Administrative views

[TODO – more on this]

The Transformation Service

The Transformation Service for OpenEAI is responsible for consuming inbound synchronization messages from one system, 'system A', and transforming them into messages that are specific to another system, 'system B'. The transformed message is then sent on to interested targets.

The RDBMS Connector Suite

The RDBMS connector suite is a suite of analysis artifacts, runtime artifacts and applications that provide general foundation to perform the following tasks:

1. Provide an analysis and runtime artifact that describes the relationship between an enterprise object and a set of database structures. i.e., "mappings" to tables from objects and mappings from objects to tables.
2. Read records from an extract file, build enterprise objects from the content of the extract(s) and publish synchronization messages based on the content. Generally, these files will be built and dropped off by legacy systems and will represent a series of transactions. The typical goal is for the transactions listed in the extracts to be applied to some other data source. For simplicity we'll call this piece of infrastructure the "**extract publisher**".
3. Consume synchronization messages from an authoritative source and build an extract file from the content of those messages (i.e., the enterprise objects contained within those messages). This extract can then be processed by a legacy system that's already built to process these types of files. We'll call this piece of infrastructure the "**extract builder**".

These components will be general infrastructure. This means they will be used as often as possible to process many different extracts, for both scenarios listed above. This infrastructure is not built with any one particular extract in mind, but rather is intended to be general enough to work for many different ones.

The File Connector Suite

Not administrable via the Console for this release. Note, even though there are no specific Console user interfaces for the File Connector Suite with this release, it is a part of the toolkit and can be run just like any other general application. In the future, there will be more specific interfaces that aid in the administration, management and running of file connections via the File Connector Suite.

The File Connector suite is a suite of analysis artifacts, runtime artifacts and applications that provide general foundation to perform the following tasks:

1. Provide an analysis and runtime artifact that describes the relationship between an enterprise object and a data extract file.
2. Read records from an extract file, build enterprise objects from the content of the extract(s) and publish synchronization messages based on the content. Generally, these files will be built and dropped off by legacy systems and will represent a series of transactions. The typical goal is for the transactions listed in the extracts to be applied to some other data source. For simplicity we'll call this piece of infrastructure the "**extract publisher**".
3. Consume synchronization messages from an authoritative source and build an extract file from the content of those messages (i.e., the enterprise objects contained within those messages). This extract can then be processed by a legacy system that's already built to process these types of files. We'll call this piece of infrastructure the "**extract builder**".

These components are general infrastructure. This means they will be used as often as possible to process many different extracts, for both scenarios listed above. This infrastructure is not built with any one particular extract in mind, but rather is intended to be general enough to work for many different ones.

The Test Suite Application

Not administrable via the Console for this release. Note, even though there are no specific Console user interfaces for the Test Suite Application with this release, it is a part of the toolkit and can be run just like any other general application. In the future, there will be more specific interfaces that aid in the administration, management and running of test suites via the test suite application.

[TODO: more information on the detailed configuration of the Test Suite]

The Message Object Generation Application

Not administrable via the Console for this release. Note, even though there are no specific Console user interfaces for the Message Object Generation Application (MOAGen) with this release, it is a part of the toolkit and can be run just like any other general application. In the future, there will be more specific interfaces that aid in the administration, management and generation of message object APIs via the Message Object Generation Application.

[TODO: more information on the detailed configuration of the MOA Gen Application]

The Console Configuration Parameters

The following table provides a list of all external configuration parameters used by the Console. The values associated to these parameters may be changed at runtime to affect the behavior of the application. Most of these parameters are currently stored in the `T_DEFAULT_CONFIG_PROPERTIES` table in the `ESB30` schema and they can be modified fia the Configuration Defaults interface within the console. However, there are three properties that are required prior to application initialization (i.e., before a database connection is established) and those properties can be found in the `web.xml` file associated to the Console web application. The location of these properties is indicated in the table below.

Important Note: The *names* of these parameters should NOT be changed. In some cases, the console looks for these specific parameter names and if they change the console will have issues running. The only thing that may have to change are the actual *values* associated to the parameters.

Parameter Name	Parameter Description	Sample Value
`applicationReport`	This parameter is used to provide the description of the "Application Report" which is one of the canned reports available via the console. This description is displayed when the user selects the type of report they want to view.	
`messageObjectReport`	This parameter is used to provide the description of the "Message Object Report" which is one of the canned reports available via the console. This description is displayed when the user selects the type of report they want to view.	
`basePrimedDocPath`	This is the path that will be used by the console to build primed document paths when associating primed documents to message objects as they are added via the console.	
`baseEoDocPath`	This is the path that will be used by the console to build enterprise object document (EO docs) paths when associating EO docs to message objects as they are added via the console.	
`contextPath`	This is the path to the console web application's "WEB-INF" directory which is used to build CLASSPATHS and derive other information.	
`appRunner`	This is the java class name of the "application runner" that is used to start applications via the console.	
`headerImagePath`	This is the path to the header image that will be displayed across the top of the web pages of the console.	
`proxyConfigDocPath`	The fully qualified path to the Proxy Service OpenEAI deployment descriptor (config doc).	
`routerConfigDocPath`	The fully qualified path to the Routing Service OpenEAI deployment descriptor (config doc).	
`elsConfigDocPath`	The fully qualified path to the Logging Service	

Parameter Name	Parameter Description	Sample Value
	OpenEAI deployment descriptor (config doc).	
generalConfigDocPath	The fully qualified path to the OpenEAI deployment descriptor (config doc) that contains all other applications managed by the console.	
xmlLayoutManager	This is the java class of the OpenEAI XML Layout manager implementation that is used when building message objects as they are added via the console.	
toolKitHome	This is the base toolkit directory (i.e. – ESB_HOME2)	
toolKitLogPath	This is the path where the console will look for log files that are created by applications being managed by the console.	
testSuiteClassName	The name of the Java class that is the test suite application. The console uses this to determine if an application is an instance of the Test Suite application and provides a link to view the Test Suite Summary document if so.	
testSuiteSummaryPathPropertyName	The name of the property associated to the Test Suite application that tells the console where to find the summary document created by the Test Suite application	
testSuiteDocUriPropertyName	The name of the property associated to the Test Suite application that tells the console where to find the actual test suite document that is input to the Test Suite application and provides the instructions to the test suite that it will execute.	
testSuitePropertyCategoryName	The name of the Property Category that contains all the test suite specific properties used by the test suite application.	
ldapHost	The LDAP host that the console will bind to when performing authentication. ***NOTE: currently this parameter is still specified in the web.xml file.***	
searchBaseDn	The base Distinguished Name (DN) that will be searched when retrieving and verifying users in the directory server. ***NOTE: currently this parameter is still specified in the web.xml file.***	
bindFilter	The directory attribute that will be appened onto the user name that the user logs in as. NOTE: this must also be specified in the jaas.conf file (or other appropriate JAAS configuration). ***NOTE: currently this parameter is still specified in the web.xml file.***	
consoleProcessIdentifier	A property that is added to the process (JVM) when an application is started so administrators can determine the applications that have been	

Parameter Name	Parameter Description	Sample Value
	started by the console externally.	
`maxProcessInitializationTime`	The amount of time (in milliseconds) that the console should wait for an application to complete initialization (startup).	
`supportEmailAddress`	The email address where email comments will be sent by the console when the user encounters an issue or when the user uses the "contact us" feature of the console.	
`mailServerName`	The SMTP mail host that will be used to send and email.	
`adminRoleName`	The name of the console administrator role. This is the role that can manage other users of the console.	
`additionalRuntimeLibPaths`	Colon or semi-colon delimited (depending on the OS) list of paths containing runtime libraries (jars and zip files) that are required by applications that may be deployed into and managed by the console.	
`linkColor`	The color that will be used for the hyperlinks that are displayed in the Headers and Footers of the Console application. This value can be changed to correspond with the header image being used if needed. Valid colors are (case sensitive): aqua, red, fuchsia, white, olive, yellow, gray, black, navy, green, teal, lime, maroon, orange, blue, purple	