http://www.openxri.org

OpenXRI Server User Manual

Version Log

Version 1 (March 18th 2008): Initial version, markus.sabadello@xdi.org
Version 2 (March 18th 2008): Some bugfixes, markus.sabadello@xdi.org
Version 3 (March 22nd 2008): Added documentation for AddGUIDCanonicalIDStage, markus.sabadello@xdi.org
Version 4 (April 28th 2009): Updated information about building OpenXRI, markus.sabadello@xdi.org

Table of Contents

1 Introduction

The goal of the OpenXRI Server (and the OpenXRI project in general) is to be a reference implementation of XRI specifications as defined by the **OASIS Extensible Resource Identifier (XRI) Technical Committee**.

As of the time of writing of this document, the latest specification relevant to OpenXRI is (1).

In particular, the goal of the OpenXRI Server is to comply with conformance requirements set forth in sections 2.6 and 2.7 of (1).

The URL of the OpenXRI web site is ***http://www.openxri.org***.

2 The Authority Resolution Server

This section explains the OpenXRI authority resolution server, its purpose, installation, configuration options and development interfaces.

2.1 Purpose

XRI authority resolution servers are the equivalent to name servers in DNS infrastructure. They are authoritative for one or more *root namespaces*, and their job is to resolve XRI *subsegments* under these *root namespaces* as requests are being made by XRI resolvers and their clients.

Therefore, if your goal is to control aspects of the XRI resolution process yourself by managing XRI subsegments and their associated authorities, the OpenXRI authority resolution server is for you.

For example, if you own the i-name **@company**, you could use OpenXRI to set up the community i-names **@company*usa** and **@company*austria**. From the perspective of your OpenXRI authority resolution server, **@company** is called a *root namespace*, and ***usa** and ***austria** are *subsegments* under the *root namespace*. Every *subsegment* has an associated *authority* with an *authority descriptor* (XRD document). Your OpenXRI authority resolution server is fully responsible for the *authorities* and associated *authority descriptors* to which the subsegments resolve. In order for XRI resolution to function correctly, the *root namespaces* – when resolved with service endpoint selection turned on and the service type input parameter set to **xri://$res*auth*($v*2.0)** – must point to the URI at which your OpenXRI authority resolution server is running. Therefore, *root namespaces* are "entry points" to your server, under which you can create an arbitrary number of community i-names.

The following is an example for an appropriate service endpoint that points XRI resolvers to your own OpenXRI authority resolution server:

```
<Service priority="10">
  <Type select="true">xri://$res*auth*($v*2.0)</Type>
  <ProviderID>xri://@company</ProviderID>
  <MediaType select="false">application/xrds+xml;trust=none</MediaType>
  <URI append="qxri" priority="2">http://www.myserver.com/resolve/</URI>
</Service>
```

Most i-brokers allow you to add and remove service endpoints in the XRD of your i-name.

See section 9 of (1) for more information about XRI authority resolution.

2.2 Features

The OpenXRI authority resolution server supports the following features:

- Recursing resolution (only for local subsegments) as defined in section 9.1.11 of (1).
- Support for synonyms (different XRIs that identify the same resource and resolve to the same authority)
- Support for multiple root namespaces, i.e. a single instance of the OpenXRI resolution server can manage community i-names under different roots.
- Auto-generation of CanonicalID elements (i-numbers)
- Auto-generation of LocalID elements
- Auto-generation of ProviderID elements
- Extensibility: The OpenXRI authority resolution server is highly modular, through its component and pipeline architecture. In addition, its components can be accessed programmatically by your external applications.

2.3 Limitations

Current limitations of the OpenXRI authority resolution server include:

- No proxy resolution. The OpenXRI authority resolution server only supports direct lookup of its own authorities.

2.4 Installation

Check out OpenXRI from the SVN repository as described here:
http://sourceforge.net/svn/?group_id=132761

Build a OpenXRI with Maven using the command

```
mvn install
```

This will place the OpenXRI binaries into a subdirectory named dist/.

Deploy the OpenXRI server .WAR file in your servlet container. Deploying a .WAR file on Tomcat is usually done by placing the file into the **webapps/** directory of Tomcat. Under the default configuration, it will be auto-deployed (i.e. decompressed into a context subdirectory). By default, the name of the .WAR file directly maps to the name of the context, which also becomes part of the URL under which the context can be accessed.

For example, if you are running Tomcat at your website **www.myserver.com**, and you place the file **openxri-server.war** into the **webapps/** directory, it will be auto-deployed to the **webapps/openxri-server** context subdirectory, and becomes available at the URL **http://www.myserver.com/openxri-server/**. Note that the path to the OpenXRI authority resolution servlet (by the default **/resolve**) is added to that URL, i.e. your authority resolution server will be running at **http://www.myserver.com/openxri-server/resolve**). Please refer to the documentation of Tomcat or your other servlet container for more information on how to deploy web applications.

2.5 Configuration

The web application's configuration file is – as is the case with all web applications – located at **WEB-INF/web.xml**. Its purpose is to configure mappings for the servlets that make up the web application, as well as to set init parameters for these servlets. Init parameters are used to configure the location of the OpenXRI server's main configuration file **server.xml**. The **WEB-INF/web.xml** file should contain the following entries, which load the OpenXRI authority resolution servlet and make it accessible at the path **/resolve** on your web server:

```
<servlet>
  <servlet-name>XRIServlet</servlet-name>
  <servlet-class>org.openxri.servlet.XRIServlet</servlet-class>
  <init-param>
    <param-name>server.config.class</param-name>
    <param-value>org.openxri.config.impl.XMLServerConfig</param-value>
  </init-param>
  <init-param>
    <param-name>server.config.file</param-name>
    <param-value>/WEB-INF/server.xml</param-value>
  </init-param>
  <load-on-startup>100</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>XRIServlet</servlet-name>
  <url-pattern>/resolve</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>XRIServlet</servlet-name>
  <url-pattern>/resolve/*</url-pattern>
</servlet-mapping>
```

Normally you should only have to make changes to that file if you want the server to run at a different path than **/resolve**.

Note that the **WEB-INF/web.xml** file's init parameters contain a pointer to the server's main configuration file, which is – by default – located at **WEB-INF/server.xml**. The following sections explain its purpose and contents.

2.5.1 Properties

The **WEB-INF/server.xml** contains a list of server-wide properties that are shared by all server components.

The relevant part of the **WEB-INF/server.xml** file typically looks like this:

```
    <properties>
         <property key="hostname" value="www.myserver.com" />
         <property key="hostport" value="80" />
```

```
            <property key="servletpath" value="/resolve/" />
        </properties>
```

It is important to make sure that these settings match the settings of your environment.

2.5.2 Components

In order for the OpenXRI authority resolution server to function correctly, implementations for all the core components must be configured in the **WEB-INF/server.xml** file. The configuration for each component includes a class name and an optional list of properties (key/value pairs).

See 2.6 for a description of components.

The relevant part of the **WEB-INF/server.xml** file typically looks like this:

```
<component interface="org.openxri.server.Server">
      <class>org.openxri.server.impl.BasicServer</class>
      <properties />
</component>
<component interface="org.openxri.urimapper.URIMapper">
      <class>org.openxri.urimapper.impl.FolderURIMapper</class>
      <properties />
</component>
<component interface="org.openxri.store.Store">
      <class>org.openxri.store.impl.db.DatabaseStore</class>
      <properties>
           <property key="hibernate.connection.url"
      value="jdbc:derby:openxri;create=true" />
           <property key="hibernate.connection.driver_class"
      value="org.apache.derby.jdbc.EmbeddedDriver" />
           <property key="hibernate.dialect"
      value="org.hibernate.dialect.DerbyDialect" />
           <property key="hibernate.show_sql"
      value="false,hibernate.connection.autoReconnect" />
           <property key="hibernate.connection.autoReconnectForPools"
      value="true" />
           <property key="hibernate.connection.is-connection-validation-
      required" value="true" />
           <property key="hibernate.transaction.factory_class"
      value="org.hibernate.transaction.JDBCTransactionFactory" />
           <property key="hibernate.current_session_context_class"
      value="thread" />
           <property key="hibernate.hbm2ddl.auto" value="update" />
      </properties>
</component>
<component interface="org.openxri.plugin.Plugin">
      <class>org.openxri.plugin.impl.NullPlugin</class>
      <properties />
```

```
</component>
```

2.5.3 Pipelines

At a minimum, the configuration file should include two pipelines (one named “create”, and one named “lookup”). It may contain several more pipelines depending on the complexity of your environment. Every pipeline configuration contains a list of 0 or more stages. Every stage configuration contains a class name and a list of key/value pairs.

See 2.7 for a description of pipelines.

A typical pipeline configuration looks like this:

```
<pipeline name="create">

      <stage>
      <class>org.openxri.pipeline.stages.AddSerialCanonicalIDStage</class>
      <properties />
      </stage>

</pipeline>

<pipeline name="lookup">

      <stage>
      <class>org.openxri.pipeline.stages.FromStoreStage</class>
      <properties />
      </stage>

      <stage>
      <class>org.openxri.pipeline.stages.AddProviderIDStage</class>
      <properties />
      </stage>

      <stage>
      <class>org.openxri.pipeline.stages.AddAuthorityResolutionSEPStage</cla
      ss>
      <properties>
           <property key="http" value="true" />
           <property key="https" value="true" />
      </properties>
      </stage>

      <stage>
      <class>org.openxri.pipeline.stages.AutoLocalIDsStage</class>
      <properties>
           <property key="excludeself" value="true" />
      </properties>
      </stage>

      <stage>
      <class>org.openxri.pipeline.stages.AddServerStatusStage</class>
           <properties>
                 <property key="code" value="100" />
                 <property key="text" value="Success" />
           </properties>
      </stage>

      <stage>
      <class>org.openxri.pipeline.stages.AddXMLElementStage</class>
      <properties>
           <property key="elementname" value="Server" />
           <property key="elementvalue" value="OpenXRI" />
      </properties>
      </stage>

</pipeline>
```

2.6 Components

The OpenXRI authority resolution server consists of several components. For each component several choices of implementations are available. It is also possible to write own implementations for components (for example, it is possible to write your own Store or URIMapper implementation if you need specialized functionality).

2.6.1 URIMapper

An URIMapper is responsible for parsing incoming requests. It extracts the following information:

- The subsegment(s) that need to be resolved.
- The root namespace to which the request applies.

URIMappers are also used for dynamically generating URIs that point back to your OpenXRI authority resolution server. This functionality is used by the AddAuthorityResolutionSEPStage (see 2.7.3.2).

2.6.1.1 SingleNamespaceURIMapper

This is a simple URIMapper implementation that is suitable if your OpenXRI authority resolution server is only used to resolve i-names in a single root namespace. It does not require the namespace to be part of the URI, therefore the URIs at which your i-names are resolved look simple and clean. This URIMapper implementation requires the root namespace to be provided as part of the configuration file.

For example, if your OpenXRI authority resolution server is installed at **http://www.myserver.com/resolve** and the SingleNamespaceURIMapper is configured with the namespace **@company**, then a request to **http://www.myserver.com/resolve/*test** will tell the server to return the XRD for the i-name **@company*test**.

To use the SingleNamespaceURIMapper, your **server.xml** file should contain the following:

```
<component interface="org.openxri.urimapper.URIMapper">
      <class>org.openxri.urimapper.impl.SingleNamespaceURIMapper</class>
      <properties>
           <property key="namespace" value="@company" />
      </properties>
</component>
```

Tutorial 1 (see 5.1) shows how to use the SingleNamespaceURIMapper in a typical real-world scenario.

2.6.1.2 FolderURIMapper

The FolderURIMapper should be used if your OpenXRI authority resolution server needs to be able to resolve i-names in multiple namespaces. In this case the namespace cannot be hardcoded in the configuration file, but instead has to be read from the URI to which a request is made. The namespace is expected to be appended to the server's URI in a folder-like pattern.

For example, if your OpenXRI authority resolution server is installed at **http://www.myserver.com/resolve**, then a request to **http://www.myserver.com/resolve/ns/@company/*test** will tell the server to return the XRD for the subsegment ***test** in the namespace **@company**, and a request to

http://www.myserver.com/resolve/ns/=name/*test will tell the server to return the XRD for the subsegment ***test** in the namespace **=name**.

To use the FolderURIMapper, your **server.xml** file should contain the following:

```
<component interface="org.openxri.urimapper.URIMapper">
      <class>org.openxri.urimapper.impl.FolderURIMapper</class>
      <properties />
</component>
```

The FolderURIMapper does not use any parameters in the configuration file.

Tutorial 2 (see 5.2) shows how to use the FolderURIMapper in a typical real-world scenario.

2.6.1.3 QueryURIMapper

Like the FolderURIMapper, the QueryURIMapper can also be used for OpenXRI installations that service multiple namespaces. The QueryURIMapper expects both the root namespace and the subsegments to resolve in the query string of the request URI (using the parameter names **ns** and **query**, respectively). Note that in general, the FolderURIMapper is preferable over the QueryURIMapper, since some XRI resolvers may append the query XRI to the path of the authority resolution URI instead of to the URI as a whole.

For example, if your OpenXRI authority resolution server is installed at **http://www.myserver.com/resolve**, then a request to **http://www.myserver.com/resolve?ns=@company&query=*test** will tell the server to return the XRD for the subsegment ***test** in the namespace **@company**, and a request to **http://www.myserver.com/resolve?ns=name &query=*test** will tell the server to return the XRD for the subsegment ***test** in the namespace **=name**.

To use the QueryURIMapper, your **server.xml** should contain the following:

```
<component interface="org.openxri.urimapper.URIMapper">
      <class>org.openxri.urimapper.impl.QueryURIMapper</class>
      <properties />
</component>
```

The QueryURIMapper does not use any parameters in the configuration file.

2.6.2 Server

The Server is the core component and responsible for building or retrieving an XRDS document. While this is not required by the architecture, the Server is expected to access the Store component for retrieving data about the authority to be resolved. The Server uses information from the URIMapper to determine the root namespace and query whose authority has to be resolved. Please refer to section 9 of (1) for more information about how XRDS documents are requested from a server.

The Server can process the following three kinds of requests:

1. It can produce an XRDS document for a root namespace and query. This is the most common case.

2. It can produce a self-describing XRDS document for a root namespace only. Please refer to section 9.1.6 of (1) for more information about self-describing XRDS documents.
3. It can produce an XRDS document for a mounted authority. "Mounting" is a feature that allows XRDS documents to be used for applications other than XRI resolution. For example, if you are running OpenXRI at **http://www.myserver.com/resolve**, and you mount an authority at the path **test**, then the XRD of that authority will be accessible under **http://www.myserver.com/resolve/test**.

The resulting XRDS document may consist of multiple XRD documents if the query consists of more than one subsegment for which your OpenXRI authority resolution server is authoritative, or if more than one authority was mounted at the requested path. Please refer to section 9.1.11 of (1) for more information about resolving multiple subsegments in one transaction, and refer to section 6 of (1) for more information about using XRDS documents in applications other than XRI resolution.

The Server component is also responsible for signing XRDS documents, in case SAML trusted resolution is requested. Please refer to section 10.2 of (1) for more information about SAML trusted resolution.

2.6.2.1 BasicServer

The BasicServer is the default Server implementation. It appropriately looks up subsegments and authorities from the store, and executes the LOOKUP pipeline (see 2.7.2). In the end, it sends the completed XRDS document back to the client.

If the LOOKUP pipeline returns a NULL XRD, the BasicServer automatically returns an XRDS document with a 222 (QUERY_NOT_FOUND) ServerStatus element.

If an exception occurs during execution of the LOOKUP pipeline, the BasicServer automatically returns an XRDS document with a 300 (TEMPORARY_FAIL) ServerStatus element.

To use the BasicServer, your **server.xml** file should contain the following:

```
<component interface="org.openxri.server.Server">
      <class>org.openxri.server.impl.BasicServer</class>
      <properties />
</component>
```

The BasicServer does not use any parameters in the configuration file.

2.6.2.2 NullServer

The NullServer returns empty XRDS documents for all requests and is intended for testing purposes only. It ignores information from the URIMapper, and it does not access the Store.

To use the NullServer, your **server.xml** file should contain the following:

```
<component interface="org.openxri.server.Server">
      <class>org.openxri.server.impl.NullServer</class>
      <properties />
</component>
```

The NullServer does not use any parameters in the configuration file.

2.6.3 Store

The Store component is responsible for persisting authorities and subsegments, as well as associated data like the authority descriptors (XRD documents). This data is normally retrieved by the first stage of the LOOKUP pipeline.

2.6.3.1 DatabaseStore

This is the default and most mature Store implementation. It persists authority and subsegment data using the Hibernate object/relational persistence service.

It supports the following features:

- Core functionality for persisting and retrieving authority and subsegment data.
- Several advanced lookup methods, for example to find local synonyms, and to construct full query XRIs that resolve to a given authority.
- Functionality for associating keys and attributes with authorities and subsegments.
- Functionality for store statistics.

To use the DatabaseStore, your **server.xml** file should contain the following:

```
<component interface="org.openxri.store.Store">
      <class>org.openxri.store.impl.db.DatabaseStore</class>
      <properties>
           <property key="hibernate.connection.url"
      value="jdbc:derby:openxri;create=true" />
           <property key="hibernate.connection.driver_class"
      value="org.apache.derby.jdbc.EmbeddedDriver" />
           <property key="hibernate.dialect"
      value="org.hibernate.dialect.DerbyDialect" />
           <property key="hibernate.show_sql"
      value="false,hibernate.connection.autoReconnect" />
           <property key="hibernate.connection.autoReconnectForPools"
      value="true" />
           <property key="hibernate.connection.is-connection-validation-
      required" value="true" />
           <property key="hibernate.transaction.factory_class"
      value="org.hibernate.transaction.JDBCTransactionFactory" />
           <property key="hibernate.current_session_context_class"
      value="thread" />
           <property key="hibernate.hbm2ddl.auto" value="update" />
      </properties>
</component>
```

The parameters to the DatabaseStore are directly passed to Hibernate and specify the configuration of the database you want to use. Please refer to the Hibernate documentation for information about these parameters.

2.6.3.2 NullStore

The NullStore does not persist any data and is intended for testing purposes only.

To use the NullStore, your **server.xml** file should contain the following:

```
<component interface="org.openxri.store.Store">
      <class>org.openxri.store.impl.NullStore</class>
      <properties />
</component>
```

The NullStore does not use any parameters in the configuration file.

2.6.4 Plugin

The Plugin component is used to handle requests to the server which the URIMapper cannot parse (i.e. invalid requests to your server).

2.6.4.1 SampleDataPlugin

This is a simple Plugin for testing purposes. It is not recommended for a production environment. It will be invoked when a request to the path **/load-sample-data** is made, relative to the OpenXRI servlet's base URL.

When it gets invoked, the store is loaded with some sample data. Specifically, XRDs for the following i-names are installed:

- =example*name1
- =example*name2
- @example*name3*name4

Attention: Configuring and invoking this plugin will delete all existing data in your Store.

To use the SampleDataPlugin, your **server.xml** file should contain the following:

```
<component interface="org.openxri.plugin.Plugin">
      <class>org.openxri.plugin.impl.SampleDataPlugin</class>
      <properties />
</component>
```

The SampleDataPlugin does not use any parameters in the configuration file.

2.6.4.2 NullPlugin

This Plugin does not handle any requests. This is useful if you want your server to simply return an error if it receives a resolution request that the URIMapper cannot interpret.

To use the NullPlugin, your **server.xml** file should contain the following:

```
<component interface="org.openxri.plugin.Plugin">
      <class>org.openxri.plugin.impl.NullPlugin</class>
      <properties />
</component>
```

The NullPlugin does not use any parameters in the configuration file.

2.6.4.3 RedirectPlugin

The RedirectPlugin redirects a client to a predefined URI. Remember that the Plugin only gets invoked when a request cannot be parsed by the URIMapper. Therefore the RedirectPlugin is suitable if you want to answer bad incoming request by redirecting the client to a fixed URI such as a custom error page.

To use the RedirectPlugin, your **server.xml** file should contain the following:

```
<component interface="org.openxri.plugin.Plugin">
      <class>org.openxri.plugin.impl.RedirectPlugin</class>
      <properties>
           <property key="uri" value="http://www.mysite.com/error.html" />
      </properties>
</component>
```

The RedirectPlugin expects a single property called **uri** in its configuration which contains the URI to redirect to.

2.7 Pipelines

A central feature of the OpenXRI authority resolution server is its pipeline architecture: The basic idea is that the server maintains two "pipelines", each of which consists of a number of "stages" in a particular order.

The purpose of a pipeline is to assemble an authority descriptor (XRD document).

Every stage on the pipeline can manipulate the XRD. The XRD produced by one stage becomes the input for the next stage, until all stages are executed and the pipeline is therefore completed.

One of the pipelines is called **CREATE**, the other **LOOKUP**.

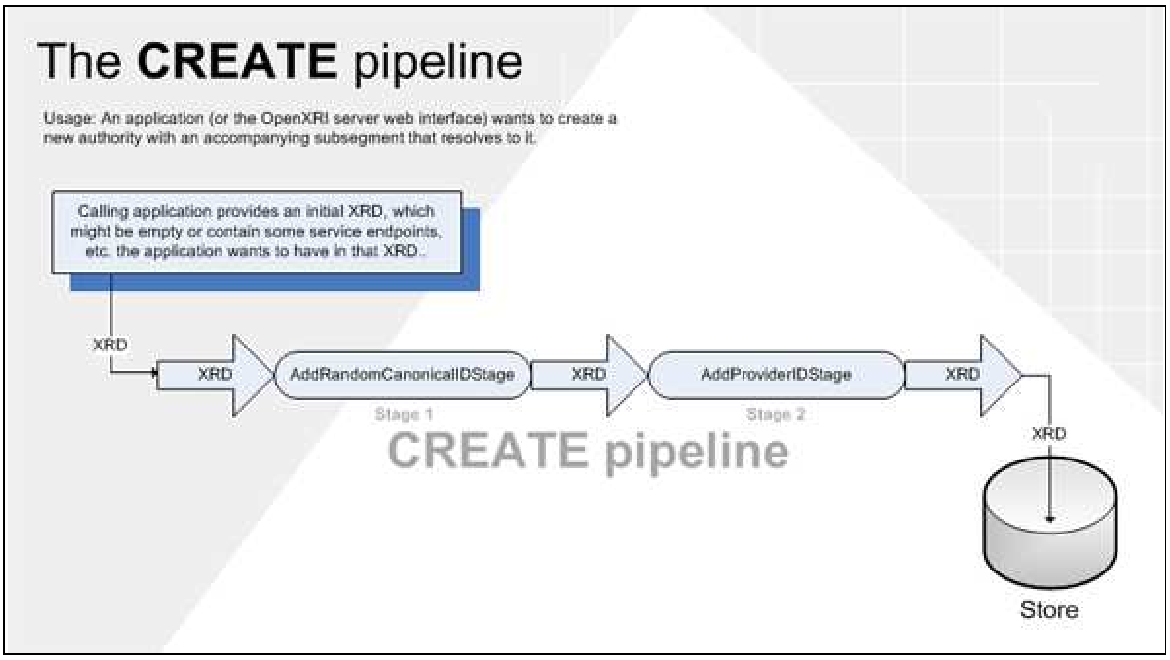
2.7.1 The CREATE Pipeline

The job of the CREATE pipeline is to assemble an XRD which is about to be persisted into the store. This pipeline is executed whenever a NEW authority (usually with an associated subsegment that resolves to it) is created.

At the beginning of this process, an initial XRD (provided by the calling application) is sent into the pipeline. This initial XRD could be empty, or contain some service endpoints, whatever data the calling application wants to associate with the new authority.

Whatever comes out at the end of the pipeline is considered to be the authority's XRD and goes into the store.

The basic idea of the CREATE pipeline is therefore to automatically set up 'static' data of an authority, i.e. data that is not expected to change between individual request, such as CanonicalIDs, service endpoints, etc.


The CREATE pipeline
Usage: An application (or the OpenXRI server web interface) wants to create a new authority with an accompanying subsegment that resolves to it.
Calling application provides an initial XRD, which might be empty or contain some service endpoints, etc. the application wants to have in that XRD.
XRD
XRD
AddRandomCanonicalIDStage
Stage 1
XRD
AddProviderIDStage
Stage 2
XRD
XRD
CREATE pipeline
Store

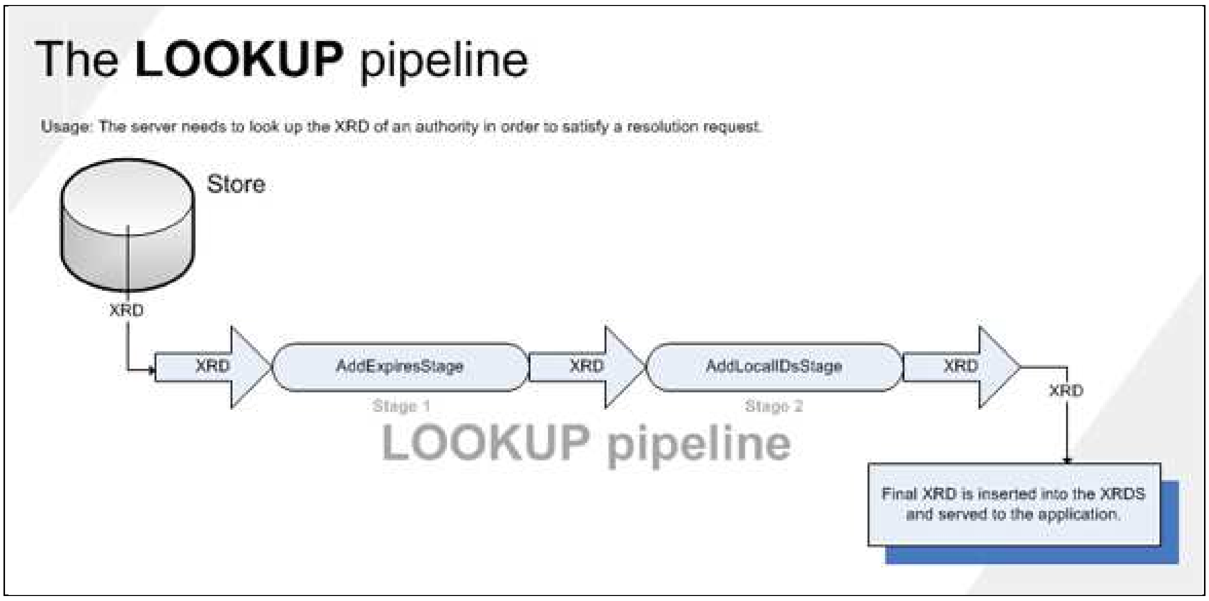
2.7.2 The LOOKUP Pipeline

The purpose of the LOOKUP pipeline is to assemble an XRD which is returned to a client resolver. This pipeline is executed whenever a resolution request for an authority has to be satisfied.

At the beginning of this process, an empty XRD is sent into the pipeline. The first stage is usually the FromStoreStage (see 2.7.3.1), which retrieves the authority's XRD from the store (i.e. the XRD that was assembled in the CREATE pipeline). Later stages on the LOOKUP pipeline then add various dynamic parts of the XRD.

Whatever comes out at the end of the pipeline is considered to be the complete XRD for the current part of the request and is added to the XRDS.

The basic idea of the LOOKUP pipeline is therefore to automatically add 'dynamic' data of a request, i.e. data that depends on the particular request, such as an <Expires> element, or an authority resolution SEP that is dynamically generated based on the server configuration.


The LOOKUP pipeline
Usage: The server needs to look up the XRD of an authority in order to satisfy a resolution request.
Store
XRD
XRD
AddExpiresStage
Stage 1
XRD
AddLocalIDsStage
Stage 2
XRD
XRD
LOOKUP pipeline
Final XRD is inserted into the XRDS and served to the application.

2.7.3 Stages

A pipeline consists of 0 or more stages. A stage can make any arbitrary modifications to the XRD that is passed into it. It could even completely replace the current XRD on the pipeline and output a fresh one. While processing an XRD, stages have access to the following information:

- The authority for which an XRD needs to be looked up (or: the parent authority of the authority for which a new XRD has to be created).
- The subsegment that resolves to this authority.
- The XRD as it currently looks like on the pipeline.
- The server's configuration including all its components and pipelines.

The stages on the CREATE and LOOKUP pipeline share the same Java interface, i.e. you can put any of them on either pipeline. Stages can also occur multiple times on a pipeline.

The following stages are included with the OpenXRI authority resolution server:

2.7.3.1 FromStoreStage

This stage loads the XRD for the current authority from the store. This is meant to be used as the first stage of the LOOKUP pipeline. This stage never makes sense on the CREATE pipeline, since at the time the CREATE pipeline is executed an authority does not have an associated XRD.

In certain exotic configurations, this stage may not even be used on the LOOKUP pipeline, i.e. if you do not need to use the Store when assembling a result XRD.

Parameters: (none)

Recommended pipeline(s) for this stage: **LOOKUP**

2.7.3.2 AddAuthorityResolutionSEPStage

This is a stage that adds an authority resolution service endpoint to the XRD. It uses the server's global properties (see 2.5.1) and the URIMapper (see 2.6.1) to appropriately construct the <URI> elements. It is recommended to put this stage into the LOOKUP pipeline, so that community i-names can be resolved to an arbitrary depth.

Parameters:

http (optional, default: true): Set to true, if a HTTP endpoint URI should be added to the service endpoint.
https (optional, default: true): Set to true, if a HTTPS endpoint URI should be added to the service endpoint.
httpport (optional, default: 80): The TCP port of the HTTP endpoint URI.
httpsport (optional, default: 443): The TCP port of the HTTPS endpoint URI.
trust (optional, default: none):The trust type to use for the service endpoint's media type element.
check-children (optional, default: true): If this is true, then the authority resolution SEP will only be added if the authority actually has child subsegments in the store.

Recommended pipeline(s) for this stage: **LOOKUP**

2.7.3.3 AddQueryStage

A stage that adds a <Query> element to the XRD and sets its value to the name of the subsegment that is being resolved.

Parameters: (none)

Recommended pipeline(s) for this stage: **LOOKUP**

2.7.3.4 AddServerStatusStage

A stage that adds a <ServerStatus> element to the XRD, indicating to the resolver the result of the resolution request.

Parameters:

code (optional, default: 100): The status code to add to the <ServerStatus> element.
text (optional, default: Success): The status text to add to the <ServerStatus> element

Recommended pipeline(s) for this stage: **LOOKUP**

2.7.3.5 AddStatusStage

A stage that adds a <Status> element to the XRD, indicating to the resolver the result of the resolution request. Note: This is meant for backwards compatibility only. According to (1), the result of the request must be indicated using a <ServerStatus> element, whereas <Status> elements are reserved for the client performing the resolution.

Parameters:

code (optional, default: 100): The status code to add to the <Status> element.
text (optional, default: Success): The status text to add to the <Status> element

Recommended pipeline(s) for this stage: **LOOKUP**

2.7.3.6 AddExpiresAbsoluteStage

A stage that adds an <Expires> tag to the XRD with an absolute expiration date.

Parameters:

expires (optional, default: 2099-01-01): The absolute expiration date to be used to construct an <Expires> tag.

Recommended pipeline(s) for this stage: **CREATE** or **LOOKUP**

2.7.3.7 AddExpiresRelativeStage

A stage that adds an <Expires> tag to the XRD with a relative expiration date.

Parameters:

seconds (optional, default: 7200): The relative expiration date (seconds from now) to be used to construct an <Expires> tag.

Recommended pipeline(s) for this stage: **CREATE** or **LOOKUP**

2.7.3.8 AddProviderIDStage

A stage that adds a <ProviderID> element to the XRD and sets its value to the parent authority's <CanonicalID>.

Parameters: (none)

Recommended pipeline(s) for this stage: **LOOKUP**

2.7.3.9 AddRandomCanonicalIDStage

A stage that adds a <CanonicalID> element to the XRD. The CanonicalID is constructed from the parent authority's CanonicalID plus a random persistent subsegment of the form !xxxx.xxxx.xxxx.xxxx.

Note that if the parent authority does not have a CanonicalID in its XRD, this stage does nothing.

Parameters: (none)

Recommended pipeline(s) for this stage: **CREATE**

2.7.3.10 AddGUIDCanonicalIDStage

A stage that adds a <CanonicalID> element to the XRD. The CanonicalID is constructed from the parent authority's CanonicalID plus a random persistent subsegment of the form of a GUID, e.g.
!xxxx.xxxx.xxxx.xxxx.xxxx.xxxx.xxxx.xxxx.

Note that if the parent authority does not have a CanonicalID in its XRD, this stage does nothing.

Parameters: (none)

Recommended pipeline(s) for this stage: **CREATE**

2.7.3.11 AddSerialCanonicalIDStage

A stage that adds a <CanonicalID> element to the XRD. The CanonicalID is constructed from the parent authority's CanonicalID plus incremental persistent subsegments (!1, !2, !3, ...).

Note that if the parent authority does not have a CanonicalID in its XRD, this stage does nothing.

Parameters: (none)

Recommended pipeline(s) for this stage: **CREATE**

2.7.3.12 AddXMLElementStage

A stage that adds an arbitrary XML element to the XRD. The text value of the element and the element's attributes can be configured using stage parameters.

Parameters:

elementname (required): The name of the element.
elementvalue (optional, default: empty string): The text inside the element.
[other parameters] (optional): Other parameters are treated as attributes for the XML element. The parameter name is the attribute name, and the parameter value is the attribute value.

Recommended pipeline(s) for this stage: **CREATE** or **LOOKUP**

2.7.3.13 AutoLocalIDsStage

A stage that automatically generates <LocalID> elements by finding synonyms in the store.

Parameters:

excludeself (optional, default: true): True, if no <LocalID> should be created for the subsegment that is being looked up.

Recommended pipeline(s) for this stage: **LOOKUP**

2.7.3.14 EmptyXRDStage

A stage that clears the whole XRD. A fresh empty XRD will be passed to the next stage.

Parameters: (none)

Recommended pipeline(s) for this stage: **CREATE** or **LOOKUP**

2.7.3.15 ExecutePipelineStage

A stage that executes a whole other pipeline as a sub-process.

pipeline (required): The name of the pipeline to execute.

Recommended pipeline(s) for this stage: **CREATE** or **LOOKUP**

2.7.3.16 InheritAttributesStage

A stage that copies attributes from the parent authority to the XRD on the pipeline

attributes (required): A comma-separated list of attributes that should be copied.

Recommended pipeline(s) for this stage: **CREATE**

2.8 Development

All components of the OpenXRI authority resolution server can be accessed programmatically from external Java applications such as servlets. This is most useful for the Store component, i.e. external applications can access the Store in order to perform functions such as registering new XRI subsegments and authorities and modifying their XRDs. In order to do so, access to the server's **WEB-INF/server.xml** file (or a copy of it) is required.

This section explains the steps necessary to get access to OpenXRI's internal components and provides several examples for common tasks.

2.8.1 The ServerConfig interface

The ServerConfig interface allows access to the server's configuration as well as its components and pipelines. An implementation for that interface can be obtained using the following code:

Option 1: Instantiation from a ServletConfig object. This is useful if you are writing a web application whose **WEB-INF/web.xml** file contains the same init parameters as the OpenXRI authority resolution server's **WEB-INF/web.xml**.

```
ServerConfig serverConfig =
      ServerConfigFactory.initSingleton(servletConfig);
```

Option 2: Instantiation with a path. This is useful if you have a ServletContext object and know the relative path to the **server.xml** file, or if you know the absolute path to the **server.xml** file (in which case the ServletContext parameter is set to null).

```
ServletContext servletContext = null;
Properties properties = new Properties();
properties.put(XMLServerConfig.SERVER_CONFIG_FILE, "path/to/server.xml");
ServerConfig serverConfig =
      ServerConfigFactory.initSingleton(servletContext, properties);
```

Note that the ServerConfig object is instantiated as a singleton. After you initialize this object, you can at any time retrieve it again using the following code:

```
ServerConfig serverConfig = ServerConfigFactory.getSingleton();
```

If your environment requires a specialized classloader in order to instantiate all of the server's components, you can use the following code, BEFORE initializing the ServerConfig object:

```
ServerConfigFactory.setClassLoader(myClassLoader);
```

2.8.2 Properties

The server-wide properties (see 2.5.1) can be accessed like this:

```
Properties properties = serverConfig.getProperties();
```

Note: It is not recommended to make changes to this Properties object. Changes will not be reflected in the underlying configuration file.

In the **WEB-INF/server.xml** file, these server-wide properties correspond to the <properties> XML element (see 2.5.1).

2.8.3 Components

In order to access the server's components (see 2.6) such as the URIMapper or the Store, an implementation of the ComponentRegistry interface must be retrieved, which then makes it possible to access the desired component. It is also possible to get the properties (key/value pairs) with which the component was initialized.

```
ComponentRegistry componentRegistry = serverConfig.getComponentRegistry();

Store store = (Store) componentRegistry.getComponent(Store.class);
URIMapper uriMapper = (URIMapper)
      componentRegistry.getComponent(URIMapper.class);

Properties storeProperties = store.getProperties();
```

It is also possible to enumerate the server's components:

```
List<Component> components = componentRegistry.getComponents();
```

Note: Never call the init() method on a component. By the time you access the component, it has already been initialized.

In the **WEB-INF/server.xml** file, these components correspond to the <component> XML elements (see 2.5.2).

2.8.4 Pipelines

In order to access the server's pipelines, an implementation of the PipelineRegistry interface must be retrieved, which then makes it possible to access the desired pipeline. Typically, at least a default CREATE and a default LOOKUP pipeline is present, however depending on your configuration there may be more pipelines that can be accessed by name.

```
PipelineRegistry pipelineRegistry = serverConfig.getPipelineRegistry();

Pipeline defaultCreatePipeline =
      pipelineRegistry.getDefaultCreatePipeline();
Pipeline defaultLookupPipeline =
      pipelineRegistry.getDefaultLookupPipeline();
Pipeline customPipeline = pipelineRegistry.getPipelineByName("MyPipeline");
```

It is also possible to enumerate the server's pipelines:

```
List<Pipeline> pipelines = componentRegistry.getPipelines();
```

The individual stages in a pipeline can also be accessed programmatically:

```
for (Stage stage : defaultCreatePipeline.getStages()) {

      System.out.println(stage.getName);
      Properties.properties = stage.getProperties();
      ...
}
```

In the **WEB-INF/server.xml** file, these pipelines correspond to the <pipeline> XML elements (see 2.5.3).

3 The Admin Interface

The OpenXRI authority resolution server comes with a web frontend for administration purposes. It provides access to the server’s configuration, its components and pipelines. The most useful feature of this web frontend is the ability to access the server’s Store component, i.e. it allows you to create new authorities and subsegments, and to edit associated data such as authority descriptors.

3.1 Installation

TODO

Check out OpenXRI from the SVN repository as described here:
http://sourceforge.net/svn/?group_id=132761

Build a .WAR file using Maven (TODO: more detailed instructions)

Deploy the .WAR file in your servlet container. Deploying a .WAR file on Tomcat is usually done by placing the file into the **webapps/** directory of Tomcat. Under the default configuration, it will be auto-deployed (i.e. decompressed into a context subdirectory). By default, the name of the .WAR file directly maps to the name of the context, which also becomes part of the URL under which the context can be accessed.

For example, if you are running Tomcat at your website **www.myserver.com**, and you place the file **openxri-admin.war** into the **webapps/** directory, it will be auto-deployed to the **webapps/openxri-admin** context subdirectory, and becomes available at the URL **http://www.myserver.com/openxri-admin/**. Please refer to the documentation of Tomcat or your other servlet container for more information on how to deploy web applications.

Security warning: The OpenXRI admin interface provides full access to the contents of your Store component, including the ability to modify and delete data. It does not include any security features that restrict access to its functionality. Security is out of scope of the OpenXRI admin interface and should be handled externally (e.g. by using Tomcat realms or Apache .htaccess files).

TODO

3.2 Configuration

The web application's configuration file is – as is the case with all web applications – located at **WEB-INF/web.xml**. Its purpose is to configure mappings for the servlets that make up the web application, as well as to set init parameters for these servlets. Init parameters are used to configure the location of the OpenXRI server's main configuration file **server.xml**.

The **WEB-INF/web.xml** file should contain the following:

```
<filter>
      <filter-name>WicketFilter</filter-name>
      <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-
      class>
      <init-param>
           <param-name>applicationFactoryClassName</param-name>
           <param-
      value>org.apache.wicket.spring.SpringWebApplicationFactory</param-
      value>
      </init-param>
      <init-param>
           <param-name>beanName</param-name>
           <param-value>wicketApplication</param-value>
      </init-param>
      <init-param>
           <param-name>server.config.class</param-name>
           <param-value>org.openxri.config.impl.XMLServerConfig</param-
      value>
      </init-param>
      <init-param>
           <param-name>server.config.file</param-name>
           <param-value>/WEB-INF/server.xml</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
</filter>
```

Normally you should not have to make changes to this file.

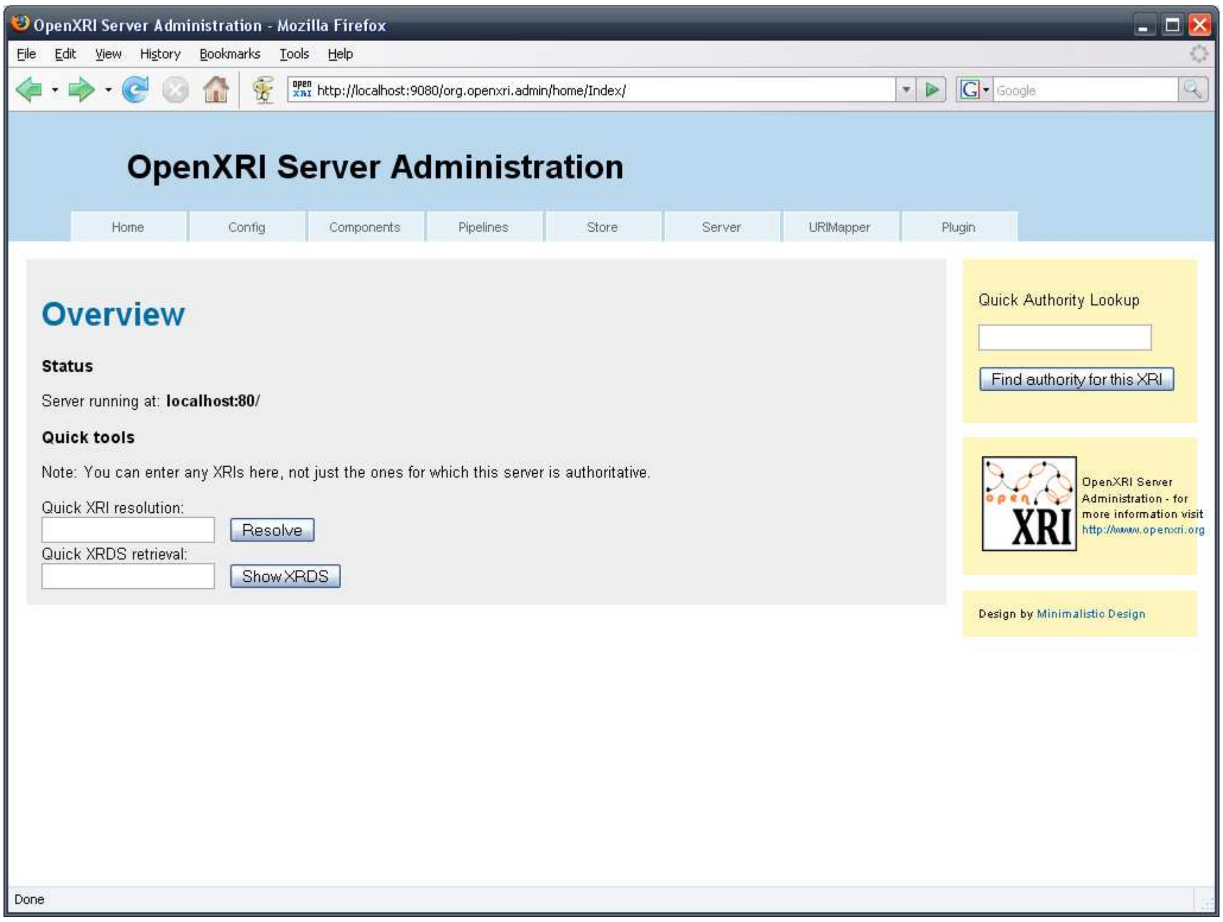
Note that the **WEB-INF/web.xml** file's init parameters contain a pointer to the server's main configuration file, which is – by default – located at **WEB-INF/server.xml**. See 2.5 for more information about this configuration file. In order for the OpenXRI admin interface to function correctly, it requires access to either the same **server.xml** file you are using for your OpenXRI authority resolution server, or to an equivalent copy of it.

3.3 The Home tab

The Home tab of the OpenXRI admin interface displays the URI of your OpenXRI authority resolution server. This information is read from the server's **server.xml** file. You should make sure that this URI is correct.

The Home tab also includes two general-purpose XRI tools. These work with any XRI, not just the ones that are resolved by your server.

- Quick XRI resolution: This will issue an HTTP redirect to the default service endpoint URI of an XRI. Using this tool with the i-name **=yourname** is equivalent to entering the following in your browser: **http://xri.net/=yourname**
- Quick XRDS retrieval: This will display the XRDS document of an XRI. Using this tool with the i-name **=yourname** is equivalent to entering the following in your browser: **http://xri.net/=yourname?_xrd_r=application/xrds+xml**


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/org.openxri.admin/home/Index/
Google
OpenXRI Server Administration
Home
Config
Components
Pipelines
Store
Server
URIMapper
Plugin
Find authority for this XRI
Resolve
Show XRDS
Done

Overview

Status

Server running at: **localhost:80/**

Quick tools

Note: You can enter any XRIs here, not just the ones for which this server is authoritative.

Quick XRI resolution:

Quick XRDS retrieval:

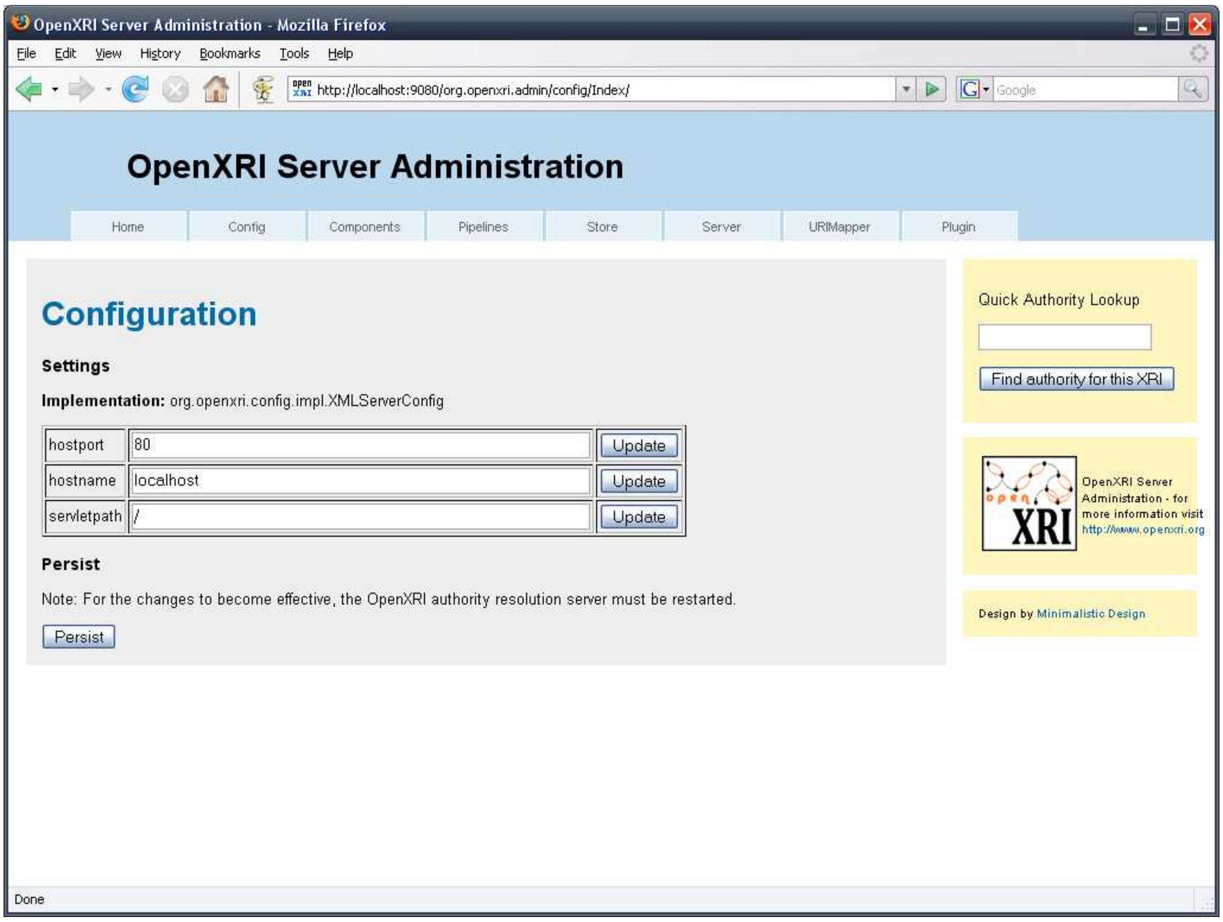
Quick Authority Lookup

OpenXRI Server Administration - for more information visit http://www.openxri.org

Design by Minimalistic Design

3.4 The Config tab

The Config tab of the OpenXRI admin interface displays the server-wide properties of your OpenXRI authority resolution server (see 2.5.1). This information is read from the server's **server.xml** file.


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/org.openxri.admin/config/Index/
OpenXRI Server Administration
Home
Config
Components
Pipelines
Store
Server
URIMapper
Plugin
Quick Authority Lookup
Find authority for this XRI
OpenXRI Server Administration - for more information visit http://www.openxri.org
Design by Minimalistic Design
Done

Configuration

Settings

Implementation: org.openxri.config.impl.XMLServerConfig

hostport	80	Update
hostname	localhost	Update
servletpath	/	Update

Persist

Note: For the changes to become effective, the OpenXRI authority resolution server must be restarted.

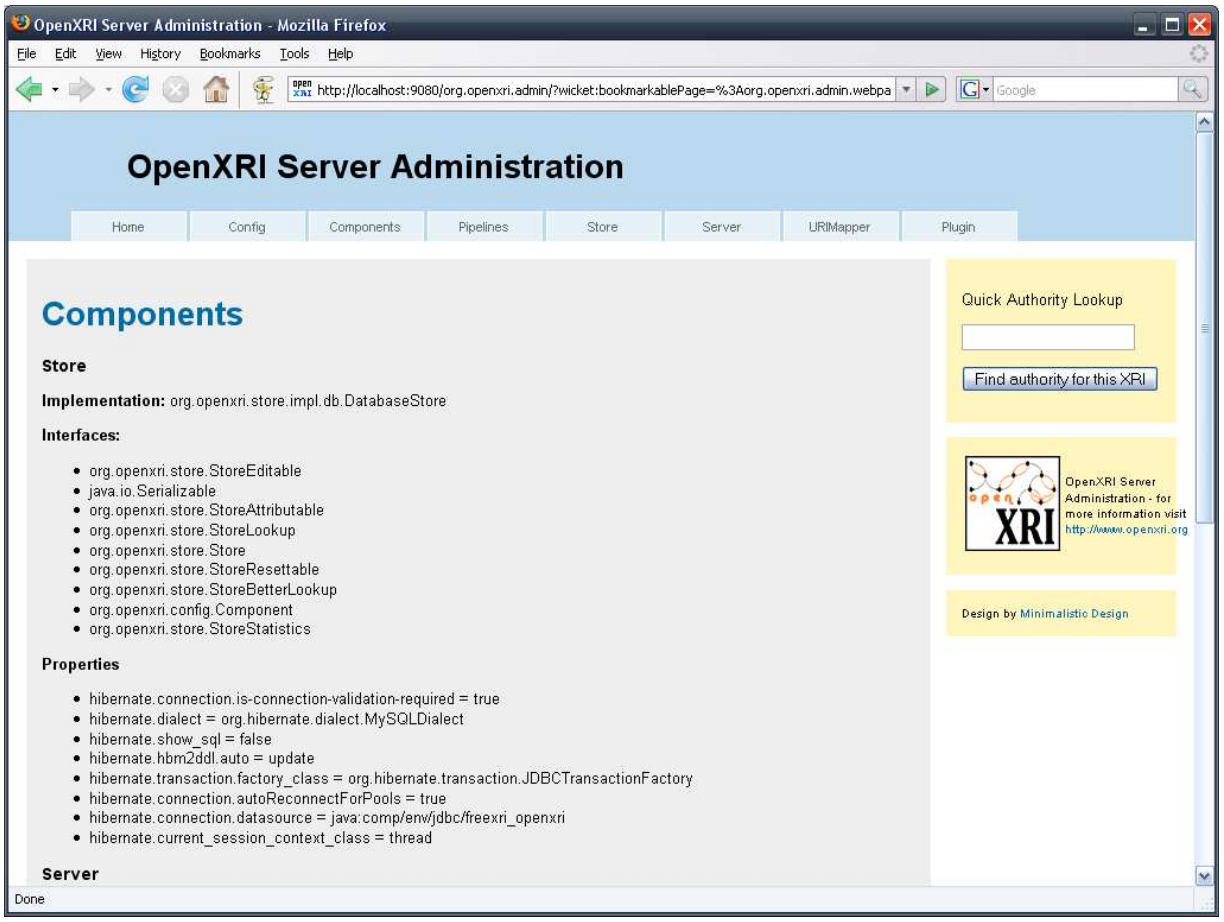
Persist

3.5 The Components tab

The Components tab of the OpenXRI admin interface displays the core components of your OpenXRI authority resolution server (see 2.5.2). This information is read from the server's **server.xml** file.

These components are: Store, Server, URIMapper and Plugin.

For each component, the implementing Java class, as well as all implemented Java interfaces are listed.


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/org.openxri.admin/?wicket:bookmarkablePage=%3Aorg.openxri.admin.webpa
OpenXRI Server Administration
Home Config Components Pipelines Store Server URIMapper Plugin
Quick Authority Lookup
Find authority for this XRI
OpenXRI Server Administration - for more information visit http://www.openxri.org
Design by Minimalistic Design
Done

Components

Store

Implementation: org.openxri.store.impl.db.DatabaseStore

Interfaces:

- org.openxri.store.StoreEditable
- java.io.Serializable
- org.openxri.store.StoreAttributable
- org.openxri.store.StoreLookup
- org.openxri.store.Store
- org.openxri.store.StoreResettable
- org.openxri.store.StoreBetterLookup
- org.openxri.config.Component
- org.openxri.store.StoreStatistics

Properties

- hibernate.connection.is-connection-validation-required = true
- hibernate.dialect = org.hibernate.dialect.MySQLDialect
- hibernate.show_sql = false
- hibernate.hbm2ddl.auto = update
- hibernate.transaction.factory_class = org.hibernate.transaction.JDBCTransactionFactory
- hibernate.connection.autoReconnectForPools = true
- hibernate.connection.datasource = java:comp/env/jdbc/freexri_openxri
- hibernate.current_session_context_class = thread

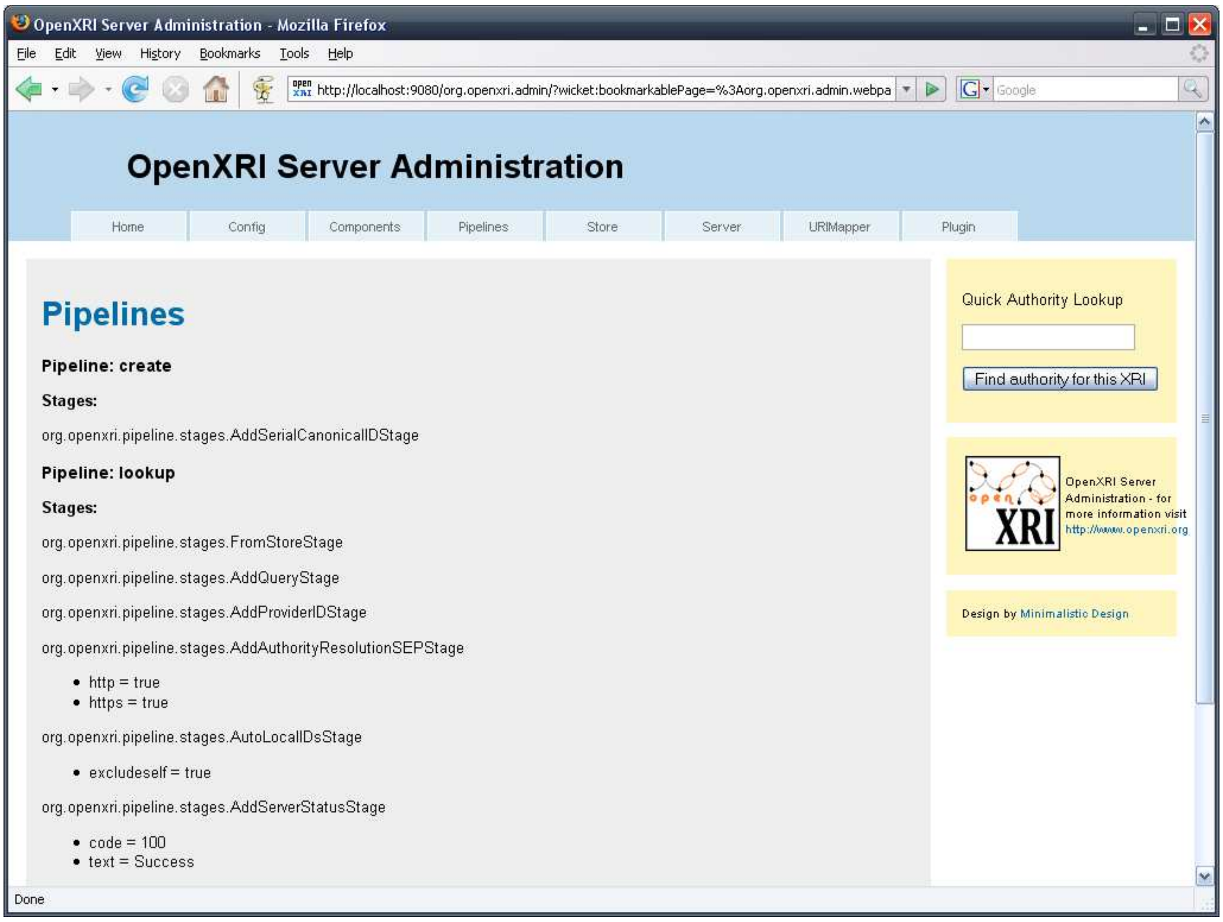
Server

3.6 The Pipelines tab

The Pipelines tab of the OpenXRI admin interface displays the pipelines of your OpenXRI authority resolution server (see 2.5.3). This information is read from the server's **server.xml** file.

There should be at least a pipeline named "create", and one named "lookup" in your configuration.

For each pipeline, its name, the stages on that pipeline, and the parameters to these stages are listed.



OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/org.openxri.admin/?wicket:bookmarkablePage=%3Aorg.openxri.admin.webpa
OpenXRI Server Administration
Home Config Components Pipelines Store Server URIMapper Plugin
Quick Authority Lookup
Find authority for this XRI
OpenXRI Server Administration - for more information visit http://www.openxri.org
Design by Minimalistic Design
Done

Pipelines

Pipeline: create

Stages:

org.openxri.pipeline.stages.AddSerialCanonicalIDStage

Pipeline: lookup

Stages:

org.openxri.pipeline.stages.FromStoreStage

org.openxri.pipeline.stages.AddQueryStage

org.openxri.pipeline.stages.AddProviderIDStage

org.openxri.pipeline.stages.AddAuthorityResolutionSEPStage

- http = true
- https = true

org.openxri.pipeline.stages.AutoLocalIDsStage

- excludeself = true

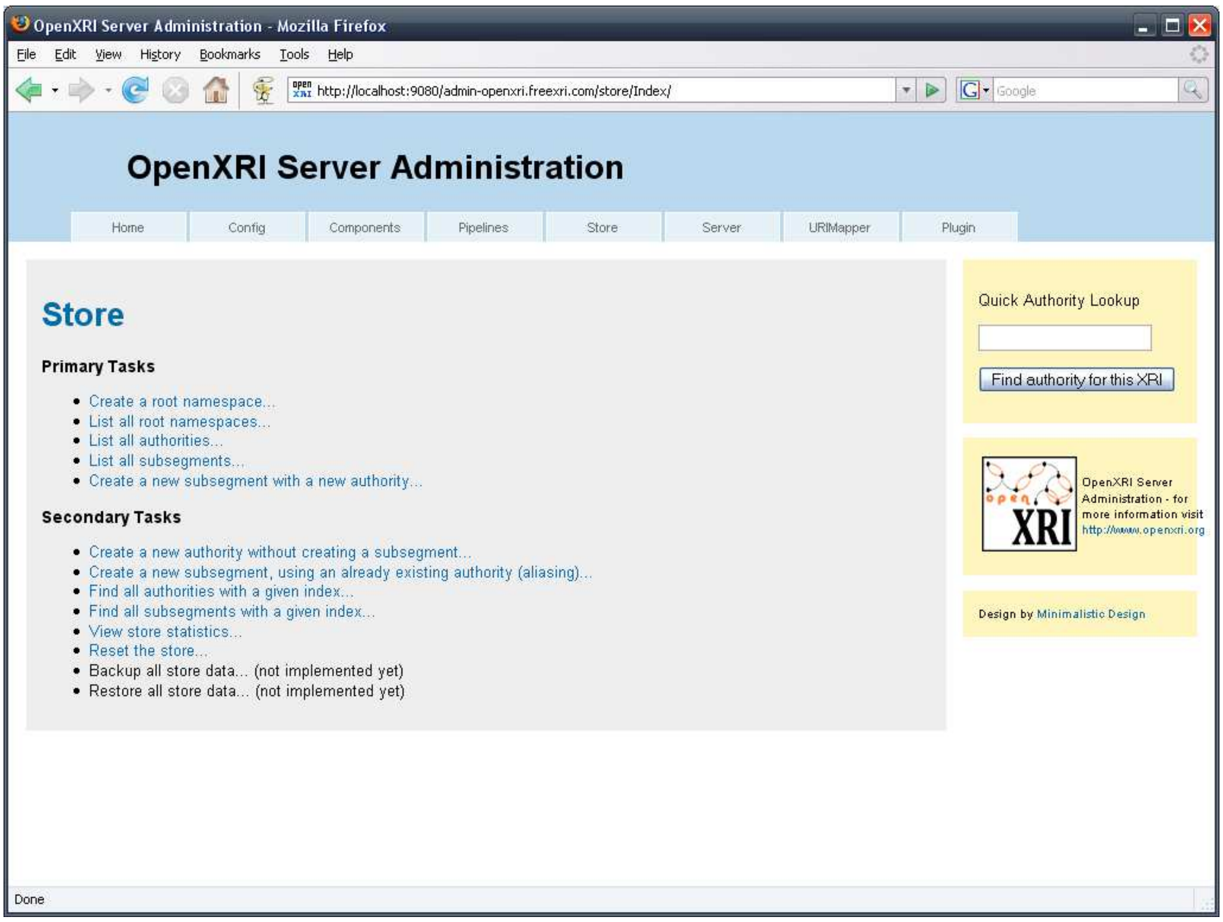
org.openxri.pipeline.stages.AddServerStatusStage

- code = 100
- text = Success

3.7 The Store tab

The Store tab of the OpenXRI admin interface provides access to the Store component of your OpenXRI authority resolution server (see 2.6.3).

This tab provides you with several functions for accessing and manipulating the Store's contents.


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/admin-openxri.freexri.com/store/Index/
OpenXRI Server Administration
Home Config Components Pipelines Store Server URIMapper Plugin
Quick Authority Lookup
Find authority for this XRI
OpenXRI Server Administration - for more information visit http://www.openxri.org
Design by Minimalistic Design
Done

Store

Primary Tasks

- Create a root namespace...
- List all root namespaces...
- List all authorities...
- List all subsegments...
- Create a new subsegment with a new authority...

Secondary Tasks

- Create a new authority without creating a subsegment...
- Create a new subsegment, using an already existing authority (aliasing)...
- Find all authorities with a given index...
- Find all subsegments with a given index...
- View store statistics...
- Reset the store...
- Backup all store data... (not implemented yet)
- Restore all store data... (not implemented yet)

3.7.1 Primary Tasks

Primary (most frequent) tasks to be performed on the Store component are:

- Create a root namespace: This will add a new root namespace to your OpenXRI authority resolution server. Root namespaces are absolute XRIs that – when resolved with service endpoint selection turned on and the service type input parameter set to **xri://$res*auth*($v*2.0)** – point to the URI at which your OpenXRI authority resolution server is running, i.e. they are "entry points" to your server, under which you can create an arbitrary number of community i-names. For example, if the i-name **@company** contains an authority resolution service endpoint that points to your server, then **@company** is a root namespace from your server's point of view.

Note that registering a root namespace in your Store automatically enables the ability to serve self-describing XRDS documents for that root namespace's authority.

- List all root namespaces: This will give you a list of all currently registered root namespaces. If you click on one of them, you will be taken to the "Subsegment Details" page for that root namespace.
- List all authorities: This will give you a list of all currently registered authorities. If you click on one of them, you will be taken to the "Authority Details" page for that authority. Note that this is not usually very useful, since the listed authorities are only identified by a number. Normally you will want to use the "Quick Authority Lookup" function instead (see).
- List all subsegments: This will give you a list of all currently registered subsegments and root namespaces. If you click on one of them, you will be taken to the "Subsegment Details" page for that subsegment. Note that this is not usually very useful, since the listed subsegments are only identified by a relative name. Normally you will want to use the "Quick Authority Lookup" function instead (see).
- Create a new subsegment: This is one of the most common tasks to be performed on the OpenXRI Store. By specifying a parent authority and a new local subsegment name it allows you to create a new subsegment and a new authority. Note that this page can also be reached from any "Authority Details" page, which is usually much more convenient, since the "Parent Authority" field will already be filled in appropriately. The authority descriptor (XRD document) for the new authority will be run through the CREATE pipeline (see 2.7.1) of your configuration before being persisted in the store. After the subsegment and authority are created, you will be taken to the "Authority Details" page for the new authority.

3.7.2 Secondary Tasks

Secondary (less frequent) tasks include:

- Create a new authority without creating a subsegment: This task allows you to create a new authority in your Store without creating any subsegment that resolves to it. Note that this is not usually useful, since your new authority will not be resolvable by any XRI, unless you explicitly create a new subsegment later that uses this authority. Another case where this task may be useful is if you want to mount the authority at a specified path, instead of using it for XRI resolution. The authority descriptor (XRD document) for the new authority will be run through the CREATE pipeline (see 2.7.1) of your configuration before being persisted in the store. After the authority is created, you will be taken to the "Authority Details" page for the new authority.
- Create a new subsegment using an existing authority: By specifying a parent authority, authority and local subsegment name, this task allows you to create a new subsegment in your Store without creating a new authority. Instead, the new subsegment will use an already existing authority. This task can be used to connect an authority that does not yet have any subsegments that resolve to it, or to connect multiple subsegments to the same authority (which is useful for local synonyms). Note that this page can also be reached from any "Subsegment Details" page, which is usually much more convenient, since the "Parent Authority" and "Authority" fields will already be filled in appropriately. After the subsegment is created, you will be taken to the "Subsegment Details" page for the new subsegment.
- Find all authorities with a given index: Authorities can have an "index", which acts like a tag by which the authority can be looked up. This task will list all authorities whose "index" field has a given value.
- Find all subsegments with a given index: Subsegment can have an "index", which acts like a tag by which the subsegment can be looked up. This task will list all subsegments whose "index" field has a given value.

- Find all authorities with a given mount path: Authorities can have a “mount path”, which makes them accessible by other applications than XRI resolution. This task will list all authorities whose “mount path” field has a given value.
- View store statistics: This will display the number of subsegments and authorities in your Store.
- Reset the store: This will delete all subsegments and authorities including all associated data from your Store. There is no undo functionality for operation.
- Backup all store data: This task allows you to save all contents of your Store to a file.
- Restore all store data: This task allows you to replace all contents of your Store with the data of a backup file.

3.7.3 Quick Authority Lookup

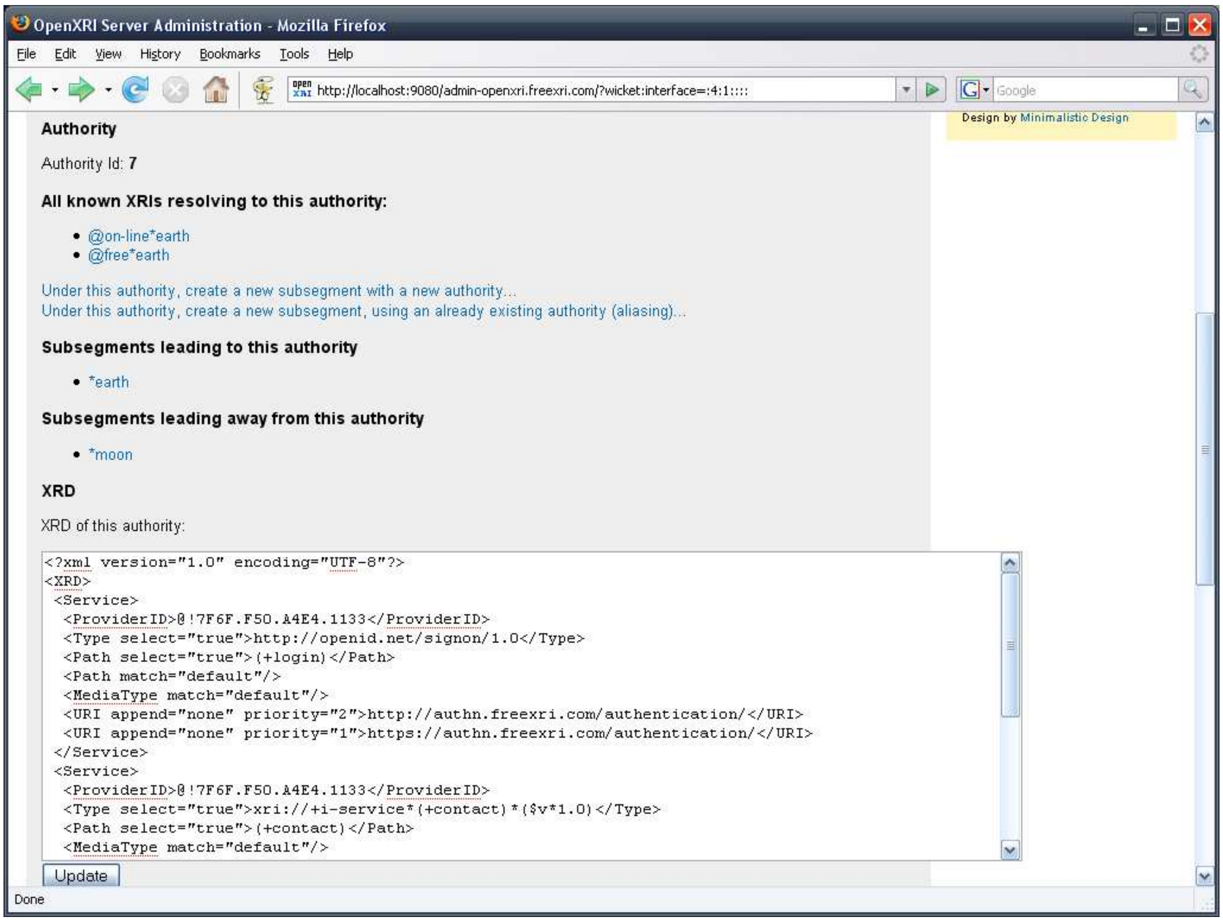
Note that the OpenXRI admin interface has a box on the right side saying “Quick Authority Lookup”. You can enter any XRI for which your OpenXRI authority resolution server is authoritative (including root namespaces), and you will be taken to the “Authority Details” page for that XRI’s authority.

3.7.4 Authority Details

The “Authority Details” page allows you to manage an authority in your Store.

It can be reached in various ways:

- By listing authorities and clicking on one of them.
- By creating a new authority.
- By using the “Quick Authority Lookup” function.


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/admin-openxri.freexri.com/?wicket:interface=:4:1::::
Design by Minimalistic Design

Authority

Authority Id: 7

All known XRIs resolving to this authority:

- @on-line*earth
- @free*earth

Under this authority, create a new subsegment with a new authority...
Under this authority, create a new subsegment, using an already existing authority (aliasing)...

Subsegments leading to this authority

- *earth

Subsegments leading away from this authority

- *moon

XRD

XRD of this authority:

```
<?xml version="1.0" encoding="UTF-8"?>
<XRD>
 <Service>
  <ProviderID>@!7F6F.F50.A4E4.1133</ProviderID>
  <Type select="true">http://openid.net/signon/1.0</Type>
  <Path select="true">(+login)</Path>
  <Path match="default"/>
  <MediaType match="default"/>
  <URI append="none" priority="2">http://authn.freexri.com/authentication/</URI>
  <URI append="none" priority="1">https://authn.freexri.com/authentication/</URI>
 </Service>
 <Service>
  <ProviderID>@!7F6F.F50.A4E4.1133</ProviderID>
  <Type select="true">xri://+i-service*(+contact)*($v*1.0)</Type>
  <Path select="true">(+contact)</Path>
  <MediaType match="default"/>
```

Update

Done

This page displays the following information about an authority:

- The authority ID (an internal identifier of that authority).
- All known XRIs resolving to this authority: This is a list of absolute XRIs that are known to resolve to this authority. This means that if you resolve one of these XRIs, the authority descriptor (XRD document) that is returned is the one associated with this authority. This list is created using information about root namespaces and subsegments in your Store.
- Subsegments leading to this authority: This is a list of subsegments that directly use this authority. Normally this is only one, unless you are using local synonyms. Clicking on one subsegment will take you to the “Subsegment Details” page.
- Subsegments leading away from this authority: This is a list of child subsegments under this authority. Clicking on one subsegment will take you to the “Subsegment Details” page.

You can make the following changes to the authority:

- Edit the authority descriptor (XRD document). You can make any changes to the XRD and update it. However, note that this is the XRD as it is persisted in the Store. When it is actually looked up by the Server, it will be run through the LOOKUP pipeline (see 2.7.2) of your configuration before being served to a client. Also note that it is not recommended to edit the <CanonicalID> element of the XRD, if there is one, as this may cause CanonicalID Verification to fail, as well as many other unpredictable side effects. Please see section 14.3 of (1) for more information about CanonicalID Verification.

- Edit attributes of the authority: Authorities can have an arbitrary number of associated key/value pairs that can be used programmatically for various purposes. You can add and remove these attributes.
- Update the index field: The index field makes it possible to programmatically look up authorities by the value of this field. You can edit and update this value. This is not required by the OpenXRI authority resolution server itself, instead this feature is meant to be used by external applications.
- Mount/Unmount the authority: This feature allows you to associate a path with the authority, which means that it will be possible to reach the authority's XRD at a URI specified by you. For example, if you are running OpenXRI at **http://www.myserver.com/resolve**, and you mount an authority at the path **test**, then the XRD of that authority will be accessible under **http://www.myserver.com/resolve/test**. This is intended for XRDS applications other than XRI resolution.
- Delete the authority: This will remove the authority including all associated data from the Store. Note that this requires you to delete all subsegment first that either lead to or away from this authority.

The "Authority Details" page also provides shortcuts to some of the tasks provided by the Store tab:

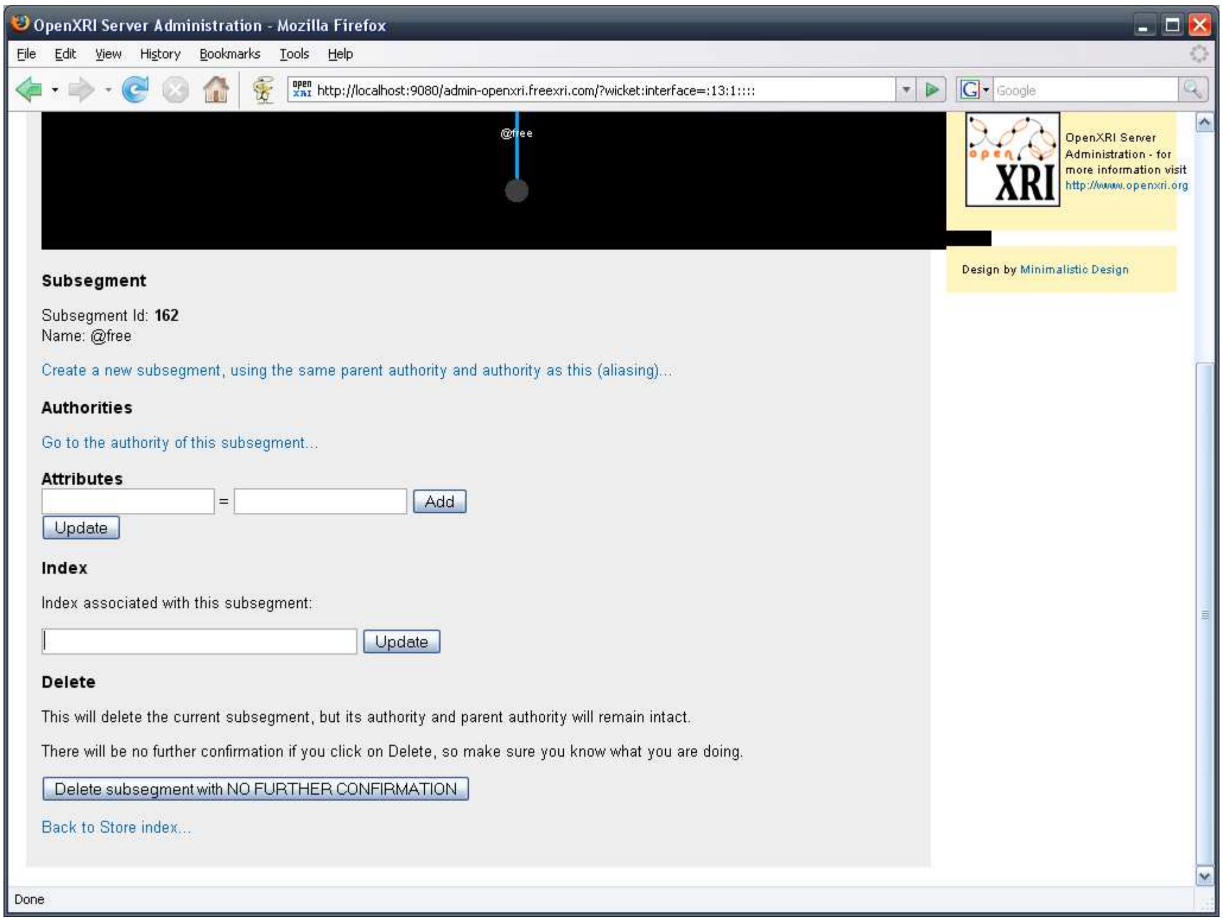
- Under this authority, create a new subsegment with a new authority: This will take you to the "Create a new subsegment" primary task, with the "Parent authority" field automatically set to the current authority.
- Under this authority, create a new subsegment, using an already existing authority: This will take you to the "Create a new authority without creating a subsegment" secondary task, with the "Parent authority" field automatically set to the current authority.

3.7.5 Subsegment Details

The "Subsegment Details" page allows you to manage a subsegment in your Store.

It can be reached in various ways:

- By listing subsegments or root namespaces and clicking on one of them.
- By creating a new subsegment.


OpenXRI Server Administration - Mozilla Firefox

http://localhost:9080/admin-openxri.freexri.com/?wicket:interface=:13:1::::

Subsegment

Subsegment Id: **162**
Name: @free

Create a new subsegment, using the same parent authority and authority as this (aliasing)...

Authorities

Go to the authority of this subsegment...

Attributes

Index

Index associated with this subsegment:

Delete

This will delete the current subsegment, but its authority and parent authority will remain intact.

There will be no further confirmation if you click on Delete, so make sure you know what you are doing.

Delete subsegment with NO FURTHER CONFIRMATION

Back to Store index...

OpenXRI Server Administration - for more information visit http://www.openxri.org

Design by Minimalistic Design

This page displays the following information about a subsegment:

- The subsegment ID (an internal identifier of that subsegment).
- The local subsegment name. This is normally a relative XRI (e.g. ***usa** or **!123**), except for root namespaces. In this case it is an absolute XRI (e.g. **@company** or **@free*newyork**).
- The authority this subsegment leads to. Clicking on “Go to the authority of this subsegment” will take you to the “Authority Details” page.
- The authority this subsegment leads away from. Clicking on “Go to the parent authority of this subsegment” will take you to the “Authority Details” page.

You can make the following changes to the subsegment:

- Edit attributes of the subsegment: Subsegments can have an arbitrary number of associated key/value pairs that can be used programmatically for various purposes. You can add and remove these attributes.
- Update the index field: The index field makes it possible to programmatically look up subsegments by the value of this field. You can edit and update this value. This is not required by the OpenXRI authority resolution server itself, instead this feature is meant to be used by external applications.
- Delete the subsegment: This will remove the subsegment including all associated data from the Store. Note that this will not delete the authorities that lead to or away from the subsegment.

The “Subsegment Details” page also provides shortcuts to some of the tasks provided by the Store tab:

- Under this authority, create a new subsegment with a new authority: This will take you to the “Create a new subsegment” primary task, with the “Parent authority” field automatically set to the current authority.
- Create a new subsegment, using the same parent authority and authority as this: This will take you to the “Create a new authority without creating a subsegment” secondary task, with the “Parent authority” and “Authority” fields automatically set to match the current subsegment. This is useful for creating local synonyms.

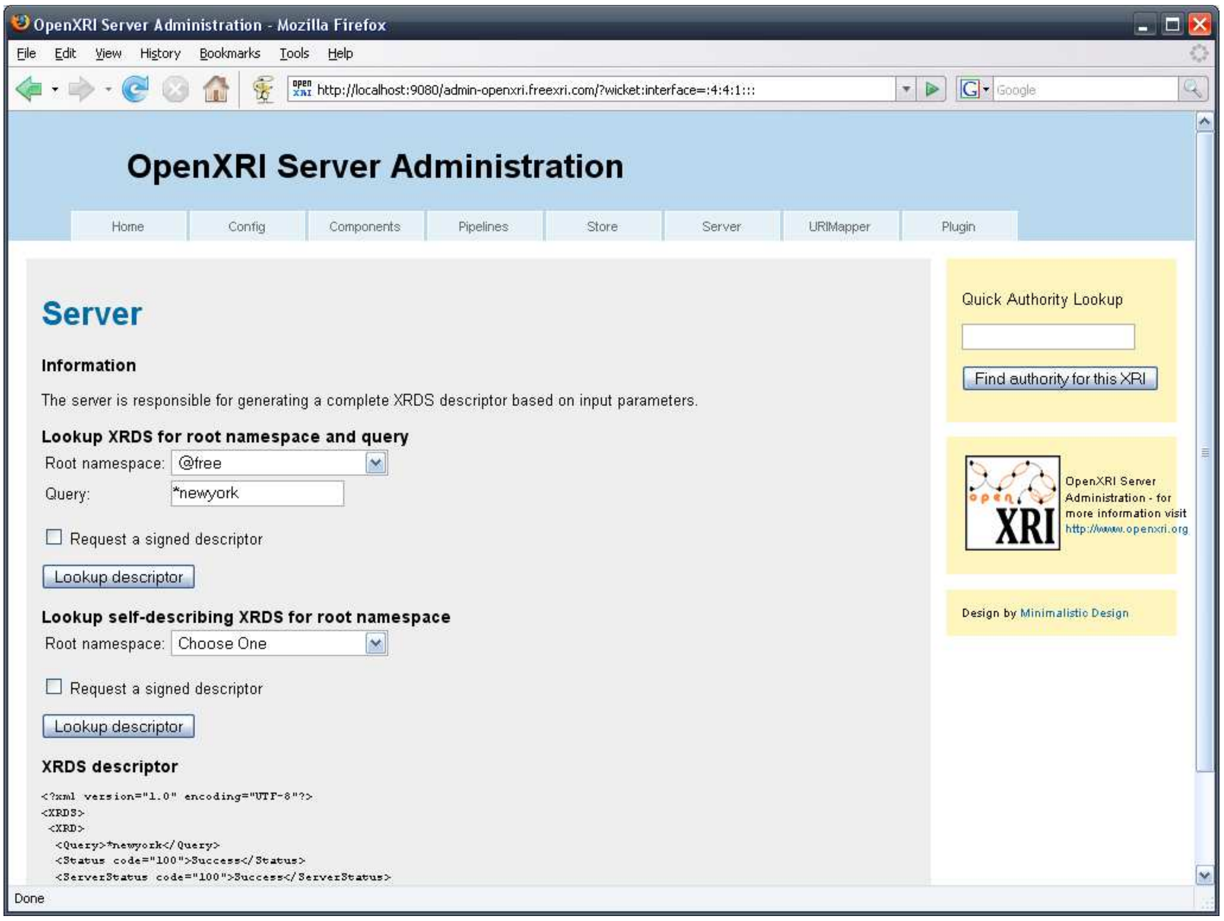
3.8 The Server tab

The Server tab of the OpenXRI admin interface provides access to the Server component of your OpenXRI authority resolution server (see 2.6.2).

It allows you to perform the following two interactive functions:

- Lookup XRDS for root namespace and query: This is the core functionality of the OpenXRI authority resolution server. Given a root namespace and query extracted by the URIMapper, the Server is able to construct a complete XRDS document. This allows you to test if your authorities and subsegments are correctly set up.
- Lookup self-describing XRDS for root namespace: Given a root namespace only, the Server is able to construct a complete self-describing XRDS document.

Both functions allow you to optionally request a signed descriptor.


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/admin-openxri.freexri.com/?wicket:interface=:4:4:1:::
Google
Home
Config
Components
Pipelines
Store
Server
URIMapper
Plugin
Quick Authority Lookup
Find authority for this XRI
open XRI
Done

OpenXRI Server Administration

Server

Information

The server is responsible for generating a complete XRDS descriptor based on input parameters.

Lookup XRDS for root namespace and query

Root namespace: @free

Query: *newyork

☐ Request a signed descriptor

Lookup descriptor

Lookup self-describing XRDS for root namespace

Root namespace: Choose One

☐ Request a signed descriptor

Lookup descriptor

XRDS descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<XRDS>
 <XRD>
  <Query>*newyork</Query>
  <Status code="100">Success</Status>
  <ServerStatus code="100">Success</ServerStatus>
```

OpenXRI Server Administration - for more information visit http://www.openxri.org

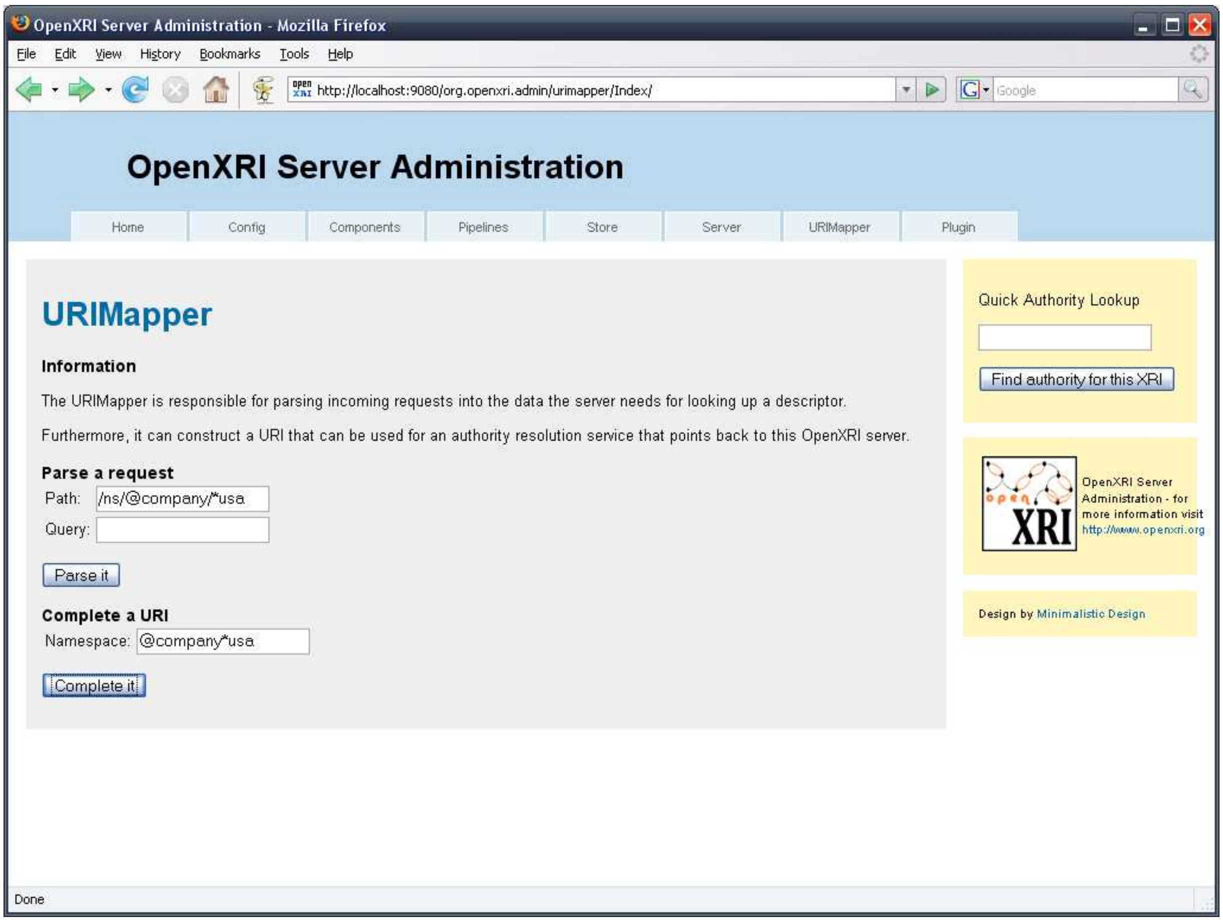
Design by Minimalistic Design

3.9 The URIMapper tab

The URIMapper tab of the OpenXRI admin interface provides access to the URIMapper component of your OpenXRI authority resolution server (see 2.6.1).

It allows you to perform the following two interactive functions:

- Parse a request: Given the path and query string of an incoming HTTP request, the URIMapper can extract a root namespace and a query. This is normally used by the BasicServer (see 0). For example, if you are using the FolderURIMapper (see 2.6.1.2) and you enter the path **/ns/@company/*usa** and an empty query string, the URIMapper should be able to extract the root namespace **@company** and the query ***usa**.
- Complete a URI: Given an XRI namespace, the URIMapper can dynamically construct a URI that points back to your OpenXRI authority resolution server. This is normally used by the AddAuthorityResolutionSEPStage (see 2.7.3.2). For example, if you enter the namespace **@company*usa**, the URIMapper should be able to construct the URI **http://www.myserver.com: 80/ resolve/ns/@company*usa/** (this will vary according to the details of your web server installation).


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/org.openxri.admin/urimapper/Index/
Google
OpenXRI Server Administration
Home Config Components Pipelines Store Server URIMapper Plugin
Quick Authority Lookup
Find authority for this XRI
OpenXRI Server Administration - for more information visit http://www.openxri.org
Design by Minimalistic Design
Done

URIMapper

Information

The URIMapper is responsible for parsing incoming requests into the data the server needs for looking up a descriptor.

Furthermore, it can construct a URI that can be used for an authority resolution service that points back to this OpenXRI server.

Parse a request

Path: /ns/@company/*usa

Query:

Parse it

Complete a URI

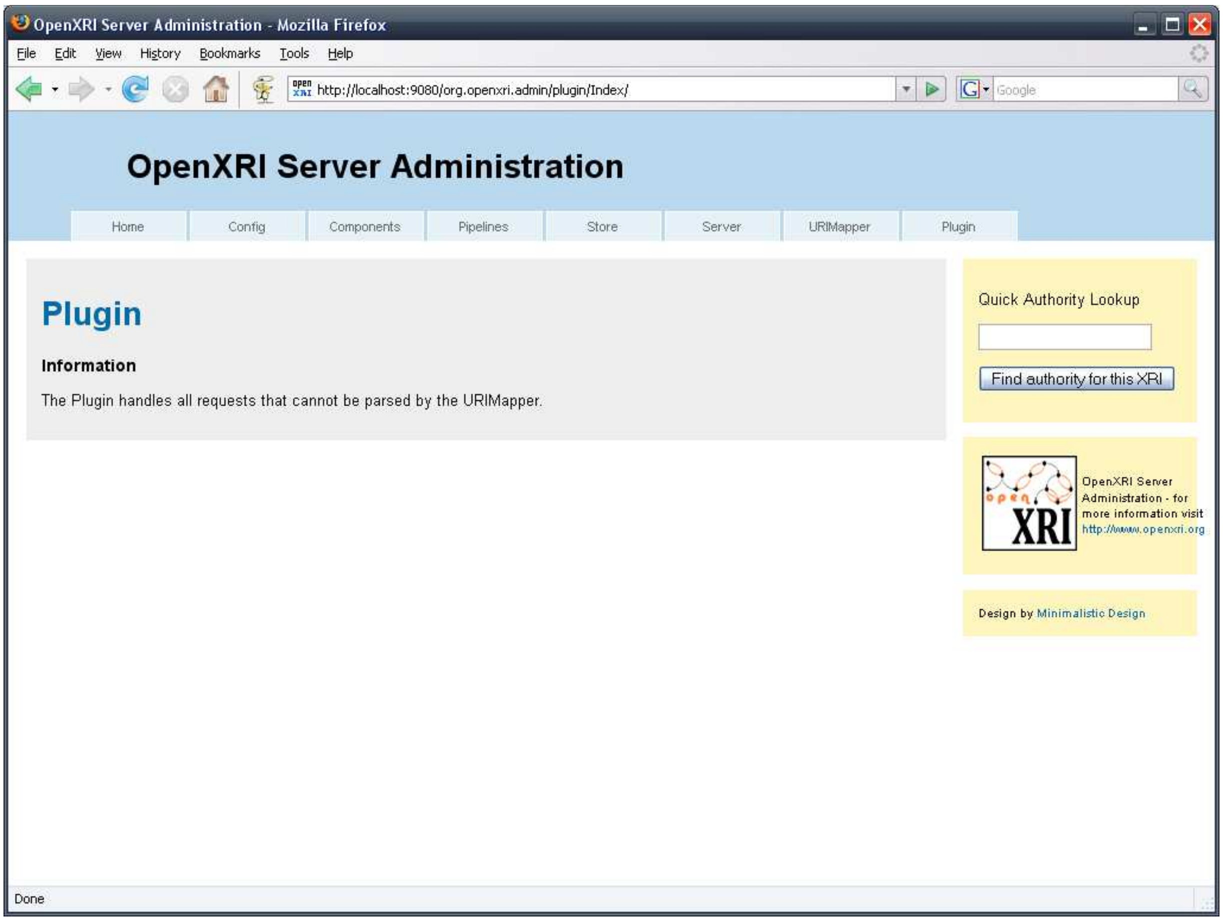
Namespace: @company*usa

Complete it

3.10 The Plugin tab

The Plugin tab of the OpenXRI admin interface displays general information about the Plugin component of your OpenXRI authority resolution server (see 2.6.4).

It does not currently provide any interactive functionality or display any information specific to the Plugin implementation in use.


OpenXRI Server Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:9080/org.openxri.admin/plugin/Index/
Google
Home
Config
Components
Pipelines
Store
Server
URIMapper
Plugin
Quick Authority Lookup
Find authority for this XRI
OpenXRI Server Administration - for more information visit http://www.openxri.org
Design by Minimalistic Design
Done

OpenXRI Server Administration

Plugin

Information

The Plugin handles all requests that cannot be parsed by the URIMapper.

4 The Proxy Resolver

This section explains the OpenXRI proxy resolver, its purpose, installation and configuration options.

4.1 Purpose

The OpenXRI proxy server can resolve any XRI on a client’s behalf. It does not depend on nor interact in any way with the OpenXRI authority resolution server.

The reference XRI proxy server runs at **http://xri.net** and is operated by the GRS (Global Registry Services). OpenXRI allows you to run your own proxy server that behaves in the same manner.

XRI proxy servers are used by clients by appending an XRI to the URL of a proxy server plus optional input parameters, for example:

http://xri.net/@free*earth*moon: This will resolve the i-name **@free*earth*moon** to the default service endpoint and issue an HTTP redirect to the URI associated with it.

http://xri.net /@free*earth?_xrd_r=application/xrds+xml: This will resolve the i-name **@free*earth** and return its complete XRDS document to the client.

Depending on input parameters, the OpenXRI proxy server can issue HTTP redirects, output XRD and XRDS documents, and perform service endpoint selection.

Please refer to (1) for more information about XRI proxy servers.

4.2 Features

- Support for HTTP Redirect, URI List, XRD and XRDS retrieval mode.
- Support for HTTP Accept: headers.
- Fully configurable XRI resolver.
- Error reporting if an XRI cannot be resolved, or if an invalid request was received.

4.3 Installation

TODO

Check out OpenXRI from the SVN repository as described here:
http://sourceforge.net/svn/?group_id=132761

Build a .WAR file using Maven (TODO: more detailed instructions)

Deploy the .WAR file in your servlet container. Deploying a .WAR file on Tomcat is usually done by placing the file into the **webapps/** directory of Tomcat. Under the default configuration, it will be auto-deployed (i.e. decompressed into a context subdirectory). By default, the name of the .WAR file directly maps to the name of the context, which also becomes part of the URL under which the context can be accessed.

For example, if you are running Tomcat at your website **www.myserver.com**, and you place the file **openxri-server.war** into the **webapps/** directory, it will be auto-deployed to the **webapps/openxri-server** context subdirectory, and becomes available at the URL **http://www.myserver.com/openxri-server/**. Note that the path to the OpenXRI proxy servlet (by the default **/proxy**) is added to that URL, i.e. your proxy server will be available at **http://www.myserver.com/openxri-server/proxy**). Please refer to the documentation of Tomcat or your other servlet container for more information on how to deploy web applications.

TODO

4.4 Configuration

The web application's configuration file is – as is the case with all web applications – located at **WEB-INF/web.xml**. Its purpose is to configure mappings for the servlets that make up the web application, as well as to set init parameters for these servlets. Init parameters are used to configure the location of the proxy's main configuration file. The **WEB-INF/web.xml** file should contain the following entries, which load the OpenXRI proxy servlet and make it accessible at the path **/proxy** on your web server:

```
<servlet>
  <servlet-name>ProxyServlet</servlet-name>
  <servlet-class>org.openxri.servlet.ProxyServlet</servlet-class>
  <init-param>
    <param-name>proxy.config.class</param-name>
    <param-value>org.openxri.config.impl.PropertiesProxyConfig</param-
    value>
  </init-param>
  <init-param>
    <param-name>proxy.properties.file</param-name>
    <param-value>/WEB-INF/proxy.properties</param-value>
  </init-param>
  <load-on-startup>100</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>/proxy</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>/proxy/*</url-pattern>
</servlet-mapping>
```

Normally you should only have to make changes to that file if you want the proxy to run at a different path than **/proxy**. See Tutorial 3 (section 5.3) for an example.

Note that the **WEB-INF/web.xml** file's init parameters contain a pointer to the proxy's main configuration file, which is – by default – located at **WEB-INF/proxy.properties**. The following section explains its purpose and contents.

4.4.1 Properties

The OpenXRI proxy server's configuration file contains a list of key/value pairs that control the operation of the proxy and the XRI resolver used by it.

A typical proxy configuration file looks like this:

```
proxy.class=org.openxri.proxy.impl.BasicProxy
MaxFollowRedirects=5
MaxFollowRefs=10
MaxRequests=15
MaxTotalBytes=10485760
MaxBytesPerRequest=1048576
SupportedResMediaTypes = application/xrds+xml, application/xrd+xml,
      text/uri-list, redirect
AtAuthority=<XRD xmlns="xri://$xrd*($v*2.0)"
      xmlns:xrd="xri://$xrd*($v*2.0)"><Service><Type>xri://$res*auth*($v*2.0
      )</Type><MediaType match="content"
      select="false">application/xrds+xml;trust=none</MediaType><URI>http://
      at.xri.net</URI></Service> </XRD>
EqualsAuthority=<XRD xmlns="xri://$xrd*($v*2.0)"
      xmlns:xrd="xri://$xrd*($v*2.0)"><Service><Type>xri://$res*auth*($v*2.0
      )</Type><MediaType match="content"
      select="false">application/xrds+xml;trust=none</MediaType><URI>http://
      equal.xri.net</URI></Service> </XRD>
BangAuthority=<XRD xmlns="xri://$xrd*($v*2.0)"
      xmlns:xrd="xri://$xrd*($v*2.0)"><Service><Type>xri://$res*auth*($v*2.0
      )</Type><MediaType match="content"
      select="false">application/xrds+xml;trust=none</MediaType><URI>http://
      bang.xri.net</URI></Service> </XRD>
```

The following list explains these settings:

- *proxy.class*: The class name of the actual Proxy implementation that answers requests
- *MaxFollowRedirects* and *MaxFollowRefs*: The maximum number of <Redirect> and <Ref> elements in XRDs that the resolver will follow while answering a single XRI resolution request
- *MaxRequests*: The maximum number of HTTP requests the resolver will make while answering a single XRI resolution request
- *MaxTotalBytes*: The maximum number of bytes the resolver will read while answering a single XRI resolution request
- *MaxBytesPerRequest*: The maximum number of bytes the resolver will read from a single HTTP request.
- *SupportedResMediaTypes*: The resolution modes the proxy will support. Possible values are:
  - *application/xrds+xml*: The proxy will resolve an XRI to an XRDS document if requested
  - *application/xrd+xml*: The proxy will resolve an XRI to an XRD document if requested
  - *text/uri-list*: The proxy will resolve an XRI to a URI list if requested
  - *redirect*: The proxy will resolve an XRI and issue a HTTP redirect. This is the default mode if the client does not specifically request a different mode.
- *AtAuthority*, *EqualsAuthority*, *BangAuthority*: These are bootstrap XRD document that act as a hints to the resolver to indicate what resolution services to use for the global context symbols @, = and !. Normally these should not be changed.

The *MaxXXX* parameters are designed to prevent the proxy from processing too complex resolution requests, which is useful as a measure against attacks on your server. The default values make sense in most environments.

5 Tutorials

This section contains three typical real-life scenarios in which OpenXRI can be used. Each tutorial is divided into a "Scenario" part, which outlines the requirements, a "Setup" part, which contains step-by-step instructions on how to fulfill these requirements, and a "Testing" part, which helps you check if everything is working as it should.

5.1 Tutorial 1: Basic community i-name resolution

5.1.1 Scenario

Let's assume you registered the top-level i-name **@company** at *http://www.2idi.com*. Your objective is to set up the three community i-names **@company*usa**, **@company*ukraine** and **@company*austria**. You own a web server at **http://www.myserver.com** which you want to use to resolve these community i-names. When resolved, their default service endpoints should point to the web site of your company's branch in the respective country, i.e. when you enter **http://xri.net/@company*usa** in your browser, you should be redirected to your company's local US site.

5.1.2 Setup

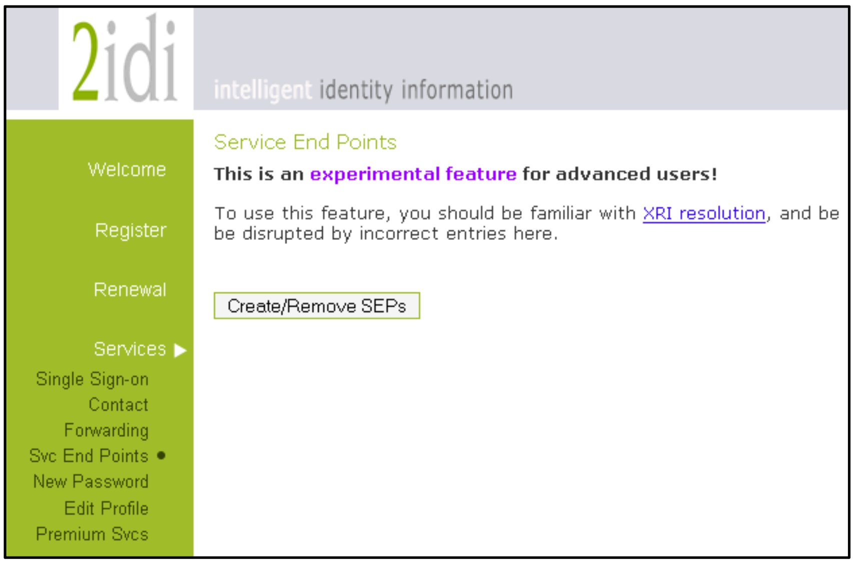
1. You need to decide at which URL you will run the OpenXRI authority resolution server. Since you own a web server at **http://www.myserver.com**, let's assume you will run OpenXRI at **http://www.myserver.com/resolve**.
2. Download and deploy the OpenXRI Server web application on your web server **www.myserver.com** as described in 2.4.
3. Download and deploy the OpenXRI admin interface web application on your web server **www.myserver.com** as described in 3.1.
4. By default, both the authority resolution server (at path **/resolve**) and the proxy server (at path **/proxy**) are activated. Since you don't need the proxy server in this tutorial, you should disable it. Your **WEB-INF/web.xml** should look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>OpenXRI Server</display-name>
  <description>OpenXRI Server</description>

  <servlet>
    <servlet-name>XRIServlet</servlet-name>
    <servlet-class>org.openxri.servlet.XRIServlet</servlet-class>
    <init-param>
      <param-name>server.config.class</param-name>
      <param-value>org.openxri.config.impl.XMLServerConfig</param-value>
    </init-param>
    <init-param>
      <param-name>server.config.file</param-name>
      <param-value>/WEB-INF/server.xml</param-value>
    </init-param>
    <load-on-startup>100</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>XRIServlet</servlet-name>
    <url-pattern>/resolve</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>XRIServlet</servlet-name>
    <url-pattern>/resolve/*</url-pattern>
  </servlet-mapping>
</web-app>
```

5. Since you want your authority resolution server to resolve only a single root namespace (**@company**), you decide to use the SingleNamespaceURIMapper for clean URIs (see 2.6.1.1). Your **server.xml** file should contain the following:

```
<component interface="org.openxri.urimapper.URIMapper">
      <class>org.openxri.urimapper.impl.SingleNamespaceURIMapper</class>
      <properties>
            <property key="namespace" value="@company" />
      </properties>
</component>
```

6. Now you need to configure your i-name **@company**. Log in at *http://www.2idi.com* with your i-name. 2idi offers functionality to edit the service endpoints of your i-name's XRD. Before you make any modifications, you may want to copy&paste your existing service endpoints to a text file, in case you want to restore them later.



Service End Points

This is an experimental feature for advanced users!

To use this feature, you should be familiar with XRI resolution, and be be disrupted by incorrect entries here.

7. If you never worked on the XRD of your i-name before, you should not have to remove any existing service endpoint. However, you have to add an authority resolution service endpoint. The purpose of this is to indicate to XRI resolvers that community i-names under your i-names will be resolved by your own OpenXRI

server. It is important that this service endpoint contains your i-name in the <ProviderID> element, and the correct URI for your OpenXRI server. It should look like this:

```
<Service priority="10">
  <Type select="true">xri://$res*auth*($v*2.0)</Type>
  <ProviderID>xri://@company</ProviderID>
  <MediaType select="false">application/xrds+xml;trust=none</MediaType>
  <URI append="qxri" priority="2">http://www.myserver.com/resolve/</URI>
</Service>
```

8. Using the OpenXRI admin interface, create the root namespace **@company** in the Store of your server. This tells your server that it is responsible for community i-names in that namespace.
9. Using the OpenXRI admin interface, create the subsegments ***usa**, ***ukraine** and ***austria** under the root namespace **@company**. From now on, the community i-names **@company*usa**, **@company*ukraine** and **@company*austria** exist and can be resolved.
10. Using the OpenXRI admin interface, edit the XRDs of the newly created authorities. Add a default service endpoint (the admin interface provides templates for service endpoints) to each of them and edit the <URI> element of that service endpoint to point to whatever URL you want your community i-name to resolve, e.g. the URL of your company's branch in the specific country.

5.1.3 Testing

1. Resolve your top-level i-name **@company** to an XRDS document. You can do this by using either a public proxy resolver (e.g. type this in your browser: **http://xri.net/@company?_xrd_r=application/xrds+xml**) or some other XRI resolution tool (like *http://www.freexri.com/tools/XRIResolution*). Make sure that the XRD of **@company** contains the authority resolution service endpoint that points to the URL of your OpenXRI authority resolution server.
2. Try manually requesting an XRDS document for your community i-name at your OpenXRI authority resolution server by entering the URL **http://www.myserver.com/resolve/*usa** in your browser. This should give you an XRDS document with a single XRD (for the subsegment ***usa**) in it.
3. Resolve your community i-name **@company*usa** to an XRDS document. You can do this by using either a public proxy resolver (e.g. type this in your browser: **http://xri.net/@company*usa?_xrd_r=application/xrds+xml**) or some other XRI resolution tool (like *http://www.freexri.com/tools/XRIResolution*). Make sure the XRD contains a default service endpoint that points to the URL of your company's branch in that country.
4. Enter **http://xri.net/@company*usa** in your browser. You should get redirected to the URI of the default service endpoint.
5. Use an XRI traceroute tool, such as the one included with the OpenXRI Client package, or the one at *http://www.freexri.com/tools/XRITraceroute*. This will show you details about each resolution step. When you use this tool with your community i-name **@company*austria**, two different servers should be involved:
   a. equals.xri.net (resolving i-names in the namespace **@**)
   b. www.myserver.com (resolving i-names in the namespace **@company*newyork**)

5.2 Tutorial 2: Multiple delegation and multiple namespaces

5.2.1 Scenario

Let’s assume you registered the free community i-name **@free*newyork** at *http://www.freexri.com*. Your objective is to set up the community i-name **@free*newyork*manhattan**. You own a web server at **http://www.myserver.com** which you want to use to resolve this community i-name.

You have a friend who is also interested in XRI. You want him to set up the community i-name **@free*newyork*manhattan*tribeca**. He owns a web server at **http://www.hisserver.com** which he wants to use to resolve this community i-name. When resolved, the default service endpoint should point to a web site containing information on New York’s TriBeCa neighborhood (such as *http://en.wikipedia/org/wiki/TriBeCa*), i.e. when you enter **http://xri.net/@free*newyork*manhattan*tribeca** in your browser, you should get redirected to that URL.

In this scenario, multiple authority resolution servers are involved. Community i-names under **@free** are resolved by *http://www.freexri.com*’s server. Community i-names under **@free*newyork** are resolved by your server. And community i-names under **@free*newyork*manhattan** are resolved by your friend’s server.

Note: The “your friend” role in this tutorial can of course also be played by you, using a second instance of the OpenXRI authority resolution server.

In the future, you are planning to resolve additional namespaces on your OpenXRI authority resolution server, and you want to configure it accordingly. For example, you want your server to be able to resolve community i-names in the namespaces **@free*newyork** and **@nyc**, but be able to distinguish between **@free*newyork*manhattan** and **@nyc*manhattan**.

5.2.2 Setup

1. You need to decide at which URL you will run the OpenXRI authority resolution server. Since you own a web server at **http://www.myserver.com**, let’s assume you will run OpenXRI at **http://www.myserver.com/resolve**. Your friend decides to run his OpenXRI authority resolution server at **http://www.hisserver.com/resolve**.
2. Download and deploy the OpenXRI Server web application on your web server **www.myserver.com** as described in 2.4. Your friend will do the same to deploy the OpenXRI Server web application on his web server **www.hisserver.com**.
3. Download and deploy the OpenXRI admin interface web application on your web server **www.myserver.com** as described in 3.1. Your friend will do the same to deploy the OpenXRI admin interface web application on his web server **www.hisserver.com**.
4. By default, both the authority resolution server (at path **/resolve**) and the proxy server (at path **/proxy**) are activated. Since you don’t need the proxy server in this tutorial, you and your friend should disable it. Your and your friend’s **WEB-INF/web.xml** should look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
```

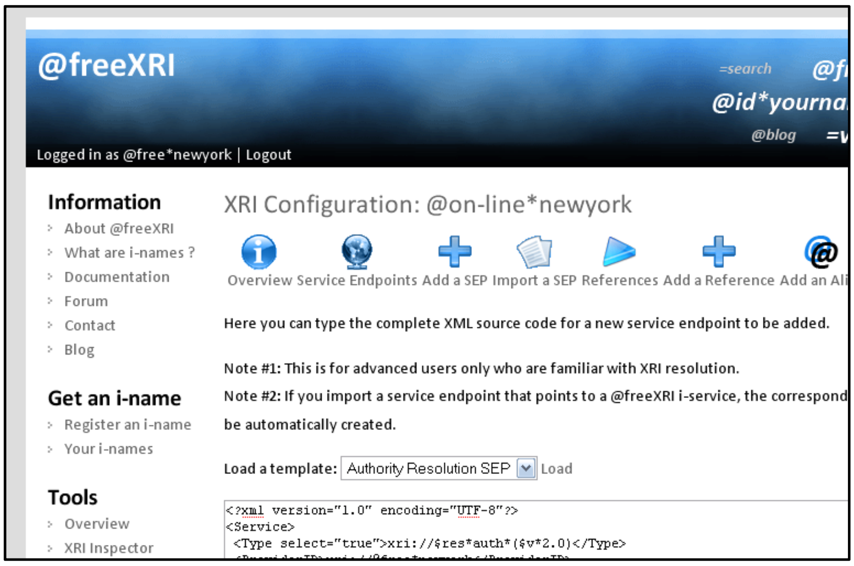
```
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>OpenXRI Server</display-name>
  <description>OpenXRI Server</description>

  <servlet>
    <servlet-name>XRIServlet</servlet-name>
    <servlet-class>org.openxri.servlet.XRIServlet</servlet-class>
    <init-param>
      <param-name>server.config.class</param-name>
      <param-value>org.openxri.config.impl.XMLServerConfig</param-value>
    </init-param>
    <init-param>
      <param-name>server.config.file</param-name>
      <param-value>/WEB-INF/server.xml</param-value>
    </init-param>
    <load-on-startup>100</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>XRIServlet</servlet-name>
    <url-pattern>/resolve</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>XRIServlet</servlet-name>
    <url-pattern>/resolve/*</url-pattern>
  </servlet-mapping>
</web-app>
```

5. Since in the future you are planning to use this server to resolve i-names in multiple namespaces, you should make the decision right from the start to use the FolderURIMapper (see section 2.6.1.2), because it can determine from the URI of an incoming request to which namespace the request is being made. Your and your friend's **server.xml** should contain this:

```
<component interface="org.openxri.urimapper.URIMapper">
      <class>org.openxri.urimapper.impl.FolderURIMapper</class>
      <properties />
</component>
```

6. Now you need to configure your i-name **@free*newyork**. Log in at *http://www.freexri.com* with your i-name. @freeXRI offers functionality to edit the service endpoints of your i-name's XRD. Before you make any modifications, you may want to copy&paste your existing service endpoints to a text file, in case you want to

restore them later.



XRI Configuration: @on-line*newyork

Here you can type the complete XML source code for a new service endpoint to be added.

Note #1: This is for advanced users only who are familiar with XRI resolution.

Note #2: If you import a service endpoint that points to a @freeXRI i-service, the correspond

be automatically created.

Load a template: Authority Resolution SEP

```
<?xml version="1.0" encoding="UTF-8"?>
<Service>
 <Type select="true">xri://$res*auth*($v*2.0)</Type>
```

7. If you never worked on the XRD of your i-name before, you should not have to remove any existing service endpoint. However, you have to add an authority resolution service endpoint. The purpose of this is to indicate to XRI resolvers that community i-names under your i-name will be resolved by your own OpenXRI server. It is important that this service endpoint contains your i-name in the <ProviderID> element, and the correct URI for your OpenXRI server. It should look like this:

```
<Service priority="10">
  <Type select="true">xri://$res*auth*($v*2.0)</Type>
  <ProviderID>xri://@free*newyork</ProviderID>
  <MediaType select="false">application/xrds+xml;trust=none</MediaType>
  <URI append="qxri"
      priority="2">http://www.myserver.com/resolve/ns/@free*newyork/</URI>
</Service>
```

Note that your i-name **@free*newyork** is encoded in the <URI> element. This is how the FolderURIMapper determines the root namespace to which a request is being made. This way your OpenXRI authority resolution server can handle an arbitrary number of root namespaces.

8. Using the OpenXRI admin interface, create the root namespace **@free*newyork** in the Store of your server. This tells your server that it is responsible for community i-names in that namespace.
9. Using the OpenXRI admin interface, create the new subsegment ***manhattan** under the root namespace **@free*newyork**. From now on, the community i-name **@free*newyork*manhattan** exists and can be resolved.
10. Using the OpenXRI admin interface, edit the XRD of the newly created authority. Add an authority resolution service endpoint to it. The purpose of this is to indicate to XRI resolvers that community i-names under your i-name will be resolved by your friend's OpenXRI server. It is important that this service endpoint contains your i-name in the <ProviderID> element, and the correct URI for your friend's OpenXRI server. It should look like this:

```
<Service priority="10">
```

```
   <Type select="true">xri://$res*auth*($v*2.0)</Type>
   <ProviderID>xri://@free*newyork*manhattan</ProviderID>
   <MediaType select="false">application/xrds+xml;trust=none</MediaType>
   <URI append="qxri"
       priority="2">http://www.hisserver.com/resolve/ns/@free*newyork*manhatt
       an/</URI>
</Service>
```

11. Using the OpenXRI admin interface, your friend creates the root namespace **@free*newyork*manhattan** in the Store of your server. This tells his server that it is responsible for community i-names in that namespace.
12. Using the OpenXRI admin interface, your friend creates the new subsegment ***tribeca** under the root namespace **@free*newyork*manhattan**. From now on, the community i-name **@free*newyork*manhattan*tribeca** exists and can be resolved.
13. Using the OpenXRI admin interface, your friend edits the XRD of the newly created authority. He adds a default service endpoint (the admin interface provides templates for service endpoints) to it and edits the <URI> element of that service endpoint to point to whatever URL he wants his community i-name to resolve, e.g. a web site containing information on New York's TriBeCa neighborhood (such as *http://en.wikipedia/org/wiki/TriBeCa*).

5.2.3 Testing

1. Resolve your i-name **@free*newyork** to an XRDS document. You can do this by using either a public proxy resolver (e.g. type this in your browser: **http://xri.net/@free*newyork?_xrd_r=application/xrds+xml**) or some other XRI resolution tool (like *http://www.freexri.com/tools/XRIResolution*). Make sure that the XRD of **@free*newyork** contains the authority resolution service endpoint that points to the URL of your OpenXRI authority resolution server.
2. Try manually requesting an XRDS document for your community i-name at your OpenXRI authority resolution server by entering the URL **http://www.myserver.com/resolve/ns/@free*newyork/*manhattan** in your browser. This should give you an XRDS document with a single XRD (for the subsegment ***manhattan**) in it.
3. Resolve your community i-name **@free*newyork*manhattan** to an XRDS document. You can do this by using either a public proxy resolver (e.g. type this in your browser: **http://xri.net/@free*newyork*manhattan?_xrd_r=application/xrds+xml**) or some other XRI resolution tool (like *http://www.freexri.com/tools/XRIResolution*). Make sure that the XRD contains the authority resolution service endpoint that points to the URL of your friend's OpenXRI authority resolution server.
4. Try manually requesting an XRDS document for your friend's community i-name at his OpenXRI authority resolution server by entering the URL **http://www.hisserver.com/resolve/ns/@free*newyork*manhattan/*tribeca** in your browser. This should give you an XRDS document with a single XRD (for the subsegment ***tribeca**) in it.
5. Resolve your friend's community i-name **@free*newyork*manhattan*tribeca** to an XRDS document. You can do this by using either a public proxy resolver (e.g. type this in your browser: **http://xri.net/@free*newyork*manhattan*tribeca?_xrd_r=application/xrds+xml**) or some other XRI resolution tool (like *http://www.freexri.com/tools/XRIResolution*). Make sure the XRD contains a default service endpoint that points to a web page on New York's TriBeCa neighborhood (for example *http://en.wikipedia.org/wiki/TriBeCa*).

6. Enter **http://xri.net/@free*newyork*manhattan*tribeca** in your browser. You should get redirected to the URI of the default service endpoint.
7. Use an XRI traceroute tool, such as the one included with the OpenXRI Client package, or the one at ***http://www.freexri.com/tools/XRITraceroute***. This will show you details about each resolution step. When you use this tool with your community i-name **@free*newyork*manhattan*tribeca**, four different servers should be involved:
    - equals.xri.net (resolving i-names in the namespace **@**)
    - resolve.freexri.com (resolving i-names in the namespace **@free**)
    - www.myserver.com (resolving i-names in the namespace **@free*newyork**)
    - www.hisserver.com (resolving i-names in the namespace **@free*newyork*manhattan**)

5.3 Tutorial 3: Proxy server

5.3.1 Scenario

Since your company performs a lot of XRI resolution, it wants to set up its own XRI proxy server, instead of using **http://xri.net**. You own a web server at **http://xri.myserver.com** which you want to use as an XRI proxy resolver.

5.3.2 Setup

1. Download and deploy the OpenXRI Server web application on your web server as described in 4.3.
2. By default, both the authority resolution server (at path **/resolve**) and the proxy server (at path **/proxy**) are activated. Since you don't need the authority resolution server in this tutorial, and you want your proxy server to run directly at **http://xri.myserver.com** without any additional path, you should disable the authority resolution servlet and change the mapping for the proxy servlet. Your **WEB-INF/web.xml** should look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>OpenXRI Server</display-name>
  <description>OpenXRI Server</description>

  <servlet>
    <servlet-name>ProxyServlet</servlet-name>
    <servlet-class>org.openxri.servlet.ProxyServlet</servlet-class>
    <init-param>
      <param-name>proxy.config.class</param-name>
      <param-value>org.openxri.config.impl.PropertiesProxyConfig</param-
      value>
    </init-param>
    <init-param>
      <param-name>proxy.properties.file</param-name>
      <param-value>/WEB-INF/proxy.properties</param-value>
    </init-param>
    <load-on-startup>100</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>ProxyServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>ProxyServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

3. Your proxy server should be running fine with the default settings. If you want to fine-tune its behavior, you may want to adjust the **WEB-INF/proxy.properties** file. See 4.4 for details.

5.3.3 Testing

1. Check if your proxy resolver can resolve an i-name and issue an HTTP redirect to the URI of the default service endpoint. Enter **http://xri.myserver.com/@company** in your browser. This should take you directly to the default URI of **@company**.
2. Check if your proxy resolver can resolve an i-name and display its XRDS document. Enter **http://xri.myserver.com/@company?_xrd_r=application/xrds+xml** in your browser. This should show an XRDS document in your browser (or prompt you to download it).
3. Check if your proxy resolver correctly treats input parameters for service endpoint resolution. Enter **http://xri.myserver.com/@company/(+contact)** in your browser. This should take you to the contact page of your i-name, even if this is not the default service.
4. For more information about proxy resolution and a list of available input parameters, see section 1 of (1).

6 Works Cited

1. **OASIS eXtensible Resource Identifier (XRI) TC.** Extensible Resource Identifier (XRI) Resolution Version 2.0 - Committee Draft 03. [Online] http://docs.oasis-open.org/xri/2.0/specs/cd03/xri-resolution-V2.0-cd-03.pdf.

2. —. Extensible Resource Identifier (XRI) Syntax V2.0. [Online] http://docs.oasis-open.org/xri/xri/V2.0/xri-syntax-V2.0-cd-01.pdf.