Repository Installation, Upgrade and Administration Guide

Introduction

The Data Repository is a software component that manages an RDF database and makes it available to other applications through a REST API, and gives end users specific views of the data. It adds role-based access control of varying granularity, transactional editing, custom treatment of ontologies and minimal/fast inference, and various administrative functions on top of the RDF database.

This page explains how it works on a host computer system, and how to install and maintain it. This page serves as an application administrator's manual for the development cycle.

Components and Layout

The data repository is installed in two ***intentionally separate*** places on the host operating system:

1. A Java Servlet *web application*, or **webapp**, installed within a Servlet container such as Apache Tomcat.
2. The repository's **home directory**, a place on the filesystem containing:
    a. The repository's **configuration properties** file
    b. Files supporting the **RDF database** (opaque to users)
    c. Files supporting the small internal RDBMS used for **user authentication**.
    d. A directory for system **log files**.

Why do we want the home directory separated from the webapp? Mainly, because the webapp is prone to getting replaced completely when a new version is deployed; typically, the servlet container unpacks a new WAR file and replaces all of the old webapp. Any data files stored there would be lost. Since the configuration and data have to persist through many incarnations of the webapp, it is much safer to keep them in a separate place in the filesystem, outside of the entire servlet container hierarchy. Also, this way the webapp needs no modification after it is installed, simplifying the procedure for the system administrator.

Another advantage to the separate location is that it gives the system administrator more flexibility to assign that directory to a location with appropriate capacity, reliability, and performance.

Command-Line Tools

The installed repository includes a set of command-line tools you will use for many of the administrative tasks. They are found in the etc/ subdirectory of the repository home directory. All of them respond to these two options:

- **`--version`** - display what released version the tool came from
- **`--help`** - display a synopsis of command args and switches

For example:

```
bash ${REPO_HOME}/etc/upgrade.sh --version
upgrade.sh from release 1.1-MS4.00 SCM revision 5422
```

Installation

Platform Requirements

- This application requires **Sun's JRE version 1.6**, e.g. "Java HotSpot(TM) SE Runtime Environment".
- The repository is a pure Java webapp and ought to run on any Java Servlet container conforming to the 2.5 version of the specification. It has only been thoroughly tested on **Apache Tomcat 6.0** and **Apache Tomcat 7.0**, however.
- The supporting utility scripts and tools require a Unix environment such as **MacOS** or **Linux**. *MS Windows is **NOT** supported*.
- Aside from the Java Servlet environment, the webapp requires a separate "home" directory, located outside of the servlet container hierarchy, to which the container's JVM has read/write access.

Prerequisites

1. **System requirements**. The current eagle-i network deployment is a reference configuration. In this deployment, eagle-i institutional servers are VMs. System requirements for these VMs are available here.
2. **Unix-like operating system**. This procedure is only valid for Unix variants like Linux, Solaris, MacOSX. To run some of the scripts you will need to have these commands installed:
    - `bash`
    - `perl`
    - `curl`
    - `awk` (surely anything that calls itself unix must have awk)
    - `tr` (seriously, is tr missing? if you are running Gentoo, install an operating system)
3. **Sun's Java JDK 1.6.0.18** (though any 1.6 version ought to work just as well).
4. **Apache Tomcat web servlet container,*version 6.0** (version 7.0 also works, but this guide refers to version 6.0), configured to run with the Java JDK in #2.

- Make sure you follow Tomcat installation and configuration instructions for the Tomcat version and Linux distribution you are using; before installing the eagle-i repository, Tomcat must be fully functional. You may want to test this by using Tomcat's manager app, which should be available athttp://localhost/manager/html/ - you will need to edit the file conf/tomcat-users.xml for defining a user and a role - see this guide: Apache Tomcat 6.0 Manager App HOW_TO
- Tomcat may be configured as a standalone web server, or be fronted by an Apache httpd server. In this guide we assume the former configuration. The latter should also work, but describing it is out of our scope.
- **Tomcat must be configured to use SSL**, see the quickstart section here: Apache Tomcat 6.0 SSL Configuration HOW-TO. Note that a production server will require a valid **SSL certificate**.
    - Make sure your certificate is properly installed by using an SSL checker, e.g. http://www.geocerts.com/ssl_checker
- Network configuration for **Tomcat to respond on standard ports 80 and 443** is required. The section **Run Tomcat on Port 80 (and 443)** under #Procedures details our preferred method. Other methods (e.g. using of Apache httpd) are possible but out of scope for this guide.
- See the #Procedures section if using Ubuntu's tomcat6 package.
    - It may be necessary to download Tomcat directly and install it manually if the version supplied by the host OS's package system is not usable. Don't hesitate to do this if it is expedient; Tomcat can run as a pure Java application in a single file hierarchy, so a manual download can work just as well (if not better) than the packaged version.

1. **Apache Derby RDBMS** installed in your Tomcat servlet container.
    - A copy of Derby is provided if you need to install it.

Scalability Limits

Note that only one instance of a Repository webapp may be run on a given home directory. This means that only one JVM and Servlet Container may access that home directory and RDF dataset at any one time. This is a restriction imposed by the Sesame triplestore.

It is not possible to "scale" performance of the repository by sharing the online RDF database among multiple machines or processes. it is possible to make periodic read-only snapshots of a database and serve them from separate machines, so long as you do not allow them to be changed.

Install and Configure Repository

Step 1. Get Repository Distribution

The repository is distributed as a single Zip file. It contains a file README which identifies the software release it was built from. It is the artifact produced by the Maven project:

```
org.eagle-i:eagle-i-repository-dist
```

Step 2. Establish the Repository Home Directory

You need to determine the repository's home directory. It may be anywhere on the system so long as it satisfies these criteria:

1. **It must be owned by the same user-id under which the servlet container (Tomcat) is executing**. The repository will automatically create files and subdirectories as needed.
2. The *host filesystem* **must** have adequate space for your anticipated RDF database, logfiles, and backup snapshots.

We will call this directory `REPO_HOME` and it will appear in commands and scripts below as `${REPO_HOME}`.
Create the repository home directory in your file system. It is useful to have a base eagle-i directory to place data and configuration used by other eagle-i applications. For example,

```
mkdir /opt/eaglei
mkdir /opt/eaglei/repo
```

If necessary (i.e if you created it using your own user-id), change ownership of the directory to the user-id under which Tomcat is running. If you followed the example above, change the ownership of the two directories using the -R option. For example, if the user-id under which Tomcat executes is *tomcat*,

```
chown -R tomcat /opt/eaglei
```

Initialize it as a variable in your shell environment. In this example (Bourne/bash shell) the repository home directory is `/opt/eaglei/repo` :

```
REPO_HOME=/opt/eaglei/repo
```

Step 3. Populate the Repository Home Directory from the Distribution

Unpack the distribution Zip archive in a directory under `/tmp`:

```
cd /tmp
unzip repository-dist.zip
```

Move the contents of the unzipped directory to your repository home directory. In this example the distribution is version `1.1-MS1.00-SNAPSHOT`

```
mv /tmp/repository-1.1-MS1.00-SNAPSHOT/* ${REPO_HOME}/.
```

List the contents of the home directory:

```
cd ${REPO_HOME}
ls
```

It should contain the subdirectories `etc/` `lib/` and `webapps/`

Step 4. Locate the Servlet Container (Apache Tomcat)

Determine the Java Servlet Container's home directory (e.g. Tomcat) which is usually dictated by your host OS. For example, it may be the 'tomcat' user's home directory, `~tomcat`.

We will call this directory `CATALINA_HOME` and it will appear in commands and scripts below as

```
${CATALINA_HOME}
```

Initialize it as a variable in your shell environment. In this example (in Bourne/bash shell) the Tomcat'shome directory is `/opt/tomcat`:

```
CATALINA_HOME=/opt/tomcat
```

Step 5. Configure Tomcat: JAVA_OPTS and System Properties

Ensure that your Tomcat server is run with the following options on its JVM. The simplest way to accomplish this is to have the environment variable `JAVA_OPTS` include those options, but each platform, distro, package etc. of Tomcat has its own mechanism for setting this variable. For example, on Fedora 14, it should be in the file `/etc/tomcat6/tomcat6.conf`. If you can't find your distribution's configuration file, you may create a file `setenv.sh` in tomcat's bin directory to add the environment variable:

```
...(ONLY DO THIS if you can't find your distribution's config file)
cd ${CATALINA_HOME}/bin
touch setenv.sh
```

Edit the configuration file (`tomcat6.conf`, `setenv.sh` or whatever your distribution uses) and add the following line:

```
JAVA_OPTS="-XX:PermSize=64M -XX:MaxPermSize=256M -Xmx1024m"
```

Add the following two system properties to file `conf/catalina.properties` under the `CATALINA_HOME` directory -- the same directory where you'll find `server.xml`. The value for both of these properties is the absolute path of the repository home directory. In this example, it is `/opt/eaglei/repo`:

```
# example
org.eaglei.repository.home = /opt/eaglei/repo
derby.system.home= /opt/eaglei/repo
```

Step 6. Install Apache Derby jars if necessary

Look in your Tomcat installation's main lib directory. If there are no files named `derby.jar` or `derby-version.jar`, you must install the Derby jars from the "scripts" distribution, e.g.

```
cp ${REPO_HOME}/lib/derby-* ${CATALINA_HOME}/lib/
```

Step 7. (OPTIONAL) Choose alternate Apache Derby implementation

Are you already running applications which use a certain Apache Derby in your servlet container? If so, set the environment variable `DERBY_HOME` as documented by Apache; if not, leave it unset and the script will use its own version of Derby (the jars in its `lib/` subdirectory):

Bourne/bash shell version:

```
....(ONLY DO THIS when ALREADY running Apache Derby!) 
export DERBY_HOME=my-derby-installation-toplevel 
```

C Shell/csh version:

```
....(ONLY DO THIS when ALREADY running Apache Derby!)
setenv DERBY_HOME my-derby-installation-toplevel 
```

NOTE: You **must** use the same version of Derby to create this initial user database as the version installed in Tomcat, so if Tomcat is already running a version of Derby, set `DERBY_HOME` to use that.

Step 8. Install the Repository

Follow this step-by-step procedure. **Before you start**, make sure the Tomcat server is **not running**.

1. Navigate to Tomcat's webapps directory. **If there exist a directory named ROOT**, move it aside. The eagle-i repository **must** be the ROOT application

   ```
   cd ${CATALINA_HOME}/webapps
   mv ROOT ROOT.original
   ```

2. Copy the repository webapp to the Tomcat webapps directory:

   ```
   cp ${REPO_HOME}/webapps/ROOT.war ${CATALINA_HOME}/webapps/.
   ```

3. Create your initial administrative user login. Think of a USERNAME and PASSWORD and substitute them for the upper case words in this command:

   ```
   bash ${REPO_HOME}/etc/prepare-install.sh USERNAME PASSWORD ${REPO_HOME}
   ```

4. Start up Tomcat.
5. Run the finish-install script, which loads the data model ontology among other things. Note that you can also give it additional options to specify a personal name and email box for the initial admin user.

   ```
   bash ${REPO_HOME}/etc/finish-install.sh USERNAME PASSWORD https://localhost:8443
   ```

   ...or, with username metadata included:

   ```
   bash ${REPO_HOME}/etc/finish-install.sh \
   -f firstname \
   -l lastname \
   -m admin@ei.edu \
   USERNAME PASSWORD https://localhost:8443
   ```

6. Run the upgrade.sh script, which preforms additional configurations.

   ```
   bash ${REPO_HOME}/etc/upgrade.sh USERNAME PASSWORD https://localhost:8443
   ```

7. Copy the file default.configuration.properties in located in {`${REPO_HOME} }}into a  file named {{configuration.properties` and edit the latter to reflect your installation. See the #Configuration section below for details on the property definitions and expected values.

8. Restart Tomcat to pick up these configuration changes. Confirm that the eagle-i repository is running by visiting the admin page (login with USERNAME and PASSWORD):

```
https://localhost:8443/repository/admin
```

Upgrade

This is the procedure to upgrade an existing repository instance to a new release of the software. **All existing configurations, data, and user accounts are preserved**. However, if the upgrade includes ontology changes there will also be an extra procedure to transform the existing data to reconcile it with ontology changes.

Before Upgrading

Get the Repository Distribution

The repository release is distributed as a single Zip file. It contains a file `README` whcih identifies the software release it was built from. It is the artifact produced by the Maven project:

```
org.eagle-i:eagle-i-repository-dist
```

Back up

It would be a wise precaution to make a backup of the current repository state so you can roll back to it in case of fatal problems with the upgrade. Follow the **Backup Procedure** in the #Procedures section to get a snapshot of the current repository contents.

Step By Step Upgrade Procedure

Note that the directory macros ${CATALINA_HOME} and ${REPO_HOME} are used in the examples here; see the **Install Procedure** above for a description of what they mean.

1. Unpack the distribution Zip archive in a directory e.g. under `/tmp`:

```
cd /tmp
unzip repository-dist.zip
```

2. Shut down your Tomcat java servlet container.
3. Delete the old repo webapp subdirectory and WAR file, since there should not be any local modifcations there. For example:

```
rm -rf ${CATALINA_HOME}/webapps/ROOT*
```

4. Save the current release files in case you have to roll back:

```
cd ${REPO_HOME}
mv etc etc.old
mv lib lib.old
mv webapps webapps.old
```

5. Copy the distribution into place (in this example the distribution is version 1.7-MS1.01) -- note there are 2 steps:

```
cp -f -rp /tmp/repository-1.7-MS1.01/* ${REPO_HOME}
cp ${REPO_HOME}/webapps/ROOT.war ${CATALINA_HOME}/webapps/.
```

6. Start up your tomcat java servlet container.
7. Run the upgrade script, substituting your admin's username and password:

```
bash ${REPO_HOME}/etc/upgrade.sh USERNAME PASSWORD https://localhost:8443
```

Watch the output of upgrade.sh very carefully! Pay particular attention to the final status and any messages beginning "WARN", they will indicate problems you MUST resolve.

8. Confirm that it worked: visit the **repo admin page**, check for new version, and then follow the link to Show Data Model Ontology versions to confirm that "loaded" and "available" versions of the ontology are the same.When running the upgrade script, there may be messages about out-of-date `NG_Internal` and `NG_Query` graphs. Most likely, these are nothing to worry about -- check the release notes. These graphs are only initialized from static files when the repository was created, and afterward they accumulate statements, so reloading a new copy of the original data is not practical. Some releases may include instructions for making changes in these graphs when upgrading from previous versions.
9. Download the data migration toolkit **that corresponds to your repository version** (in this example, version 1.7-MS1.02) and run the data migration script, substituting your admin's username and password:

```
wget -O ${REPO_HOME}/etc/eagle-i-datatools-datamanagement.jar \
http://infra.search.eagle-i.net:8081/nexus/content/repositories/\
releases/org/eagle-i/eagle-i-datatools-datamanagement/1.7-MS1.02/\
eagle-i-datatools-datamanagement-1.7-MS1.02.jar

bash ${REPO_HOME}/etc/data-migration.sh -u USERNAME -p PASSWORD -r https://localhost:8443
```

Watch the output of data-migration.sh very carefully! Pay particular attention to the final status and any messages beginning "WARN", they will indicate problems you MUST resolve. In addition to the output on screen, the data-migration script will place a data migration report in the `logs` directory directly under `/etc`.

Configuration

URIs for Creating New Roles, Transitions, and Workspaces

When you create a new **Role** or **Workflow Transition**, you have the option of assigning your own URI to the new resource. When should you make up a URI, and when should you just let the system create one?

The answer is, if you expect to be exporting and sharing this resource -- which is to be expected for most Roles and Transitions, since there will typically be many commonly-administered repositories sharing the same configuration of Roles and workflow, **make up your own URIs following guidelines here**. This ensures that when, e.g. a User is copied from one repository to another, her Roles are all available on the destination repository with the same access grants. Likewise, Workflow Transitions should be given the same uniform URI on all repository sites to ensure that a change on the master site is propagated correctly. Since you ensure the Transition's URI is globally unique, you can import it on all the slave repos with the URI preserved, replacing the local copy, since the local URI will be the same as the master's URI.

For **Workspace** (aka Named Graph) URIs, you have to assign them in the process of creating a new Named Graph. Follow the rules below to create a reasonable URI.

Rules of Creating Your Own URIs

Note that these URIs **do not need to be resolvable**. They are purely *symbolic* names for instances buried within the repository, which are virtually guaranteed never to appear in the outside world. So don't worry about whether the URI is actually resolved, most of the existing URIs for these types of things are not resolvable anyway.

1. Choose a **namespace** (URI prefix):
    a. Unique to your project, e.g. http://dartmouse.edu/repo/
    b. Borrow the Repository's namespace: http://eagle-i.org/ont/repo/1.0/**(but be careful of the suffix you pick!)**
2. Choose a unique symbolic suffix:
    a. It **must not** contain the slash (/) character!
    b. Include a description of the content to prevent collisions, e.g. "Role" or "WFT"
    c. If you are borrowing the Repository namespace, start your suffix with a unique leader, e.g. "DARTMOUSE_".

Examples of good URIs:

http://dartmouse.edu/repo/Role_LabRathttp://dartmouse.edu/repo/WFT_13_2http://eagle-i.org/ont/repo/1.0/DARTMOUSE_ROLE_PIhttp://eagle-i.org/ont/repo/1.0/DARTMOUSE_WFT_TRASH
Exception: The URI of a named graph representing an ontology is usually the same as the URI of the ontology itself, i.e. the subject of its `owl:versionInfo` statement. If you should happen to add a new ontology named graph to the repository, use that URI for its name. However this should be a very rare occurrence; usually new ontological information is simply added to the existing eagle-i data model ontology graph.

Managing Access Controls on Contact & "Hidden" Properties

The repository has a mechanism for restricting access to some of the properties of resource instances, deemed "hidden" and "contact" properties - these are two distinct sets of properties, configured independently but by an identical mechanism. See the **Resource Property Hiding** and **Acces Control** sectio ns under **Concepts** in the **Repository Design Specification / API Manual** for more details about how this works.

To configure access control, bring up the Admin GUI home page, and click on the link Manage Property Access Controls under **Administrator Tasks**. This page lets you edit the Access Control List (ACL) of both contact and hiding property sets. Granting **READ** permission allows a user or role to see those properties in Dissemination and harvest reports.

It is best to grant these permissions only to Roles - there should be no need to grant property read access at the granularity of users.

Note that if you grant **READ** to the **Anonymous** pseudo-role, that is the same as turning off all protection since unauthenticated users will be able to see the hidden/contact properties.

Once you have set up a single repository to your liking, you can export and re-import the grants to other repositories. See the **Procedure: Exporting and Importing Property Access Controls** section below.

Configuration Reference

This section lists everything that can be configured, so you can get familiar with it before installing anything.

System Properties

The repository requires these system properties to be defined in the JVM environment running your servlet container:

1. `org.eaglei.repository.home` - absolute path of the *repository home directory* (see below)
2. `derby.system.home` - directory containing Derby databases.

We recommend you set to the same path as repository home

If you are using the Apache Tomcat version 6 container (which is recommended), you can add these system properties to file `conf/catalina.properties` - add lines like these: (note that the path `/opt/eaglei/repo` is just shown as an example)

```
org.eaglei.repository.home = /opt/eaglei/repo
derby.system.home= /opt/eaglei/repo
```

Repository Home Directory

The repository has a notion of a *home directory*, the root of a hierarchy of other runtime files.

We recommend that you place this home directory outside of the Java Servlet container, especially outside of the webapp structure -- this is because the entire webapp directory tree may be removed and replaced when the webapp is updated. The default location is under the process-owning user's home directory.

The path of the home directory is determined:

1. If the Java system property `org.eaglei.repository.home` is set, its value must be the absolute path of the home directory.
2. Otherwise it defaults to the user's home directory (value of system property user.home) followed by path elements eaglei and repository, e.g. `/home/lcs/eaglei/repository/`

These files and subdirectories are found under the repository home:

- `configuration.properties` - java properties file with repository and log4j configuration props. This is optional, it must be created by the administrator.
- `logs/` - Default subdirectory for log files, see configuration. Created automatically by default.
- `sesame/` - Default Sesame RDF database files - DO NOT TOUCH. Created automatically by default.
- `etc/` - Contains scripts and tools for the repo administrator.
- `db/` - Default subdirectory Derby RDBMS files - DO NOT TOUCH. Created automatically by default.

The Repository Configuration Properties File

The configuration file is read by Apache Commons Configuration, which recognizes interpolated property and system property values. See its documentation for more information about features in the configuration file.

You can set the following properties in the `configuration.properties` file. Most of the repository's "configuration" comes from administrative metadata in its RDF database and from the ontologies loaded into it, so the configuration settings here are very minimal and mostly serve to bootstrap the RDF repository.

The properties in red are **required**; those in orange are **important** and can be considered required for a production system, although they can be elided for a test or development system at the cost of some ugliness in the UI.

Any properties not present (or commented-out) in the configuration properties file will revert to the default values documented here. In most cases this is just fine. The property is only provided so that the application's behavior can be customized and adjusted to suit the requirements of a particular installation site. For example, your site may have a convention of writing all log files to a fielsystem separate from the applications.

- `eaglei.repository.namespace` - The namespace URI prefix for Eagle-I resource instances created in the repository.
  - **Every administrator should set this to a reasonable value for his/her site, because the default is NOT desireable.**
  - **The value must be a fully qualified, resolvable, HTTP URL.**
  - For example, `http://foo.bar.edu/i/`
  - Use the **http** scheme, *NOT* **https**, since the container will redirect to https if necessary, but it is not possible to direct back if it becomes preferable to use http later.
  - The system-generated default is the hostname followed by /i/ -- but this is often wrong, since Java's determination of hostnames in a servlet container environment is not reliable.
- `eaglei.repository.title` - the decorative title for UI pages, should be set for cosmetic reasons.
  - Set this to the name of your site, e.g. "Miskatonic University School of Medicine".
- `eaglei.repository.logo` - URL of the logo image for your site, may be either relative URL (to refer to a image embedded in the webapp) or an absolute URL to use an image hosted elsewhere. It should be about 50 pixels high and a suitable with given the proportions.
- `eaglei.repository.index.url` - Set this to the URL to which you want the site's "root" (top-level index) page redirected. Although the repository is installed as the root webapp to have control over resolving Semantic Web URIs, it does not need the root page so this allows you to configure your site as you like.
- `eaglei.repository.admin.backgroundColor` - Lets you change the background color for admin web UI pages, to give admins an obvious cue when they are operating on e.g. the production vs. test repos. Value is CSS color expression, e.g. crayon name like "bisque" or hex #CCFFCC (Added in Release 1.2MS2 or 3)
- `eaglei.repository.instance.xslt` - path to XSL stylesheet used to transform the HTML output of the instance dissemination service. A value for this key is **required** to produce XHTML in the dissemination service; without it, the service returns the internal XML document describing the instance.
  - If it is a relative path then it must be located relative to the root of the web application, if absolute then it is in the filesystem at large.
  - The advantage of keeping your stylesheets external to the webapp is that you can change them easily, and don't have to modify the webapp from its default installation.
  - An example is provided at `repository/styles/example.xsl` which creates very simple HTML, as a demonstration of how to write an XSL stylesheet.
- `eaglei.repository.instance.css` - URI of the CSS stylesheet resource to be used to style instance dissemination pages. It must be an absolute path or absolute URL. The default is:

```
eaglei.repository.instance.css = /repository/styles/i.css
```

- `eaglei.repository.tbox.graphs` - a comma-separated list of graph URIs making up the "TBox".
  You should <u>never</u> have to set this! It is configurable "just in case", and for testing/experimenting. For more information, see the section on **infer encing** in the API Manual.
  By default, the TBox consists of:
  - The repository's internal ontology, http://eagle-i.org/ont/repo/1.0/
  - The eagle-i data model ontology, `http://purl.obolibrary.org/obo/ero.owl`
- `eaglei.repository.datamodel.source` - the full name of a resource within the webapp which its itself a property file describing the RDF data model ontology. *You should not need to set this, the default is adequate for the eagle-i applicaiton*. Default is `eaglei-datamodel.properties` which is a built-in resource file.
  For a description of the contents of this properties file, see the separate document **Guide to Data Model Configuration Properties**
- `eaglei.repository.sesame.dir` - directory where Sesame RDF database files are created.
  - Defaults to `sesame` subdirectory of home dir.
- `eaglei.repository.log.dir` - Directory where log files are created.
  - Defaults to `logs` subdirectory of the home dir.
  - You can also configure `log4j` explicitly by adding `log4j` properties to this file.
- `eaglei.repository.sesame.indexes` - index configuration for Sesame triple store. Must be a comma-separated list of index specifiers, see Sesame NativeStore configuration documentation for details. Use this to change the internal indexes Sesame maintains to process queries. It takes effect on next servlet container (tomcat) restart.

> **WARNING**
>
> If you have a configured value and wish to go back to the default, **do NOT** just delete this configuration property. If you do, Sesame will simply keep the existing indexes. You must change it to the original default value, which is documented int he default configuration file.

- `eaglei.repository.slow.query` - Value in seconds of time after which a SPARQL query should be considered "slow" and logged as such. Only affects the SPARQL Protocol endpoint service. Default is 0, which never logs. Use this to check for performance problems, since it logs the full text of the query and time of occurance in the regular log at INFO level.
- `eaglei.repository.sparqlprotocol.max.time` - Time limit, in seconds, of the maximum time allowed for a query invoked by the SPARQL Protocol endpoint. Note that this *does not* affect any internally-generated SPARQL queries.
  - Any user can override this setting to impose a ***shorter*** timeout by giving a value for the nonstandard **time** argument.
  - Only the Administrator can override with a ***longer*** timeout.
  - The built-in default is **600 seconds** (10 min) if nothing is configured.
  - If a SPARQL Protocol request cannot be complted within the timeout, it returns an **HTTP 413 status** (result too large - it was the standard response code that comes closest to the concept).
- `eaglei.repository.anonymous.user` - **This is a hack, only intended for testing the Anonymous role**. Its value is a username, e.g. "nobody". If configured, when the designated user logs in, their session is downgraded to the **Anonymous** role; this allows explicit testing of **Anon**

ymous (vs. **Authenticated**) access even when the webapp configuration does not allow unauthenticated access. **ONLY TESTERS SHOULD EVER NEED TO SET THIS.**

- **Configuring Contact Hiding:*The following properties control the contact hiding extension, which restricts the display of "contact location" properties of instances and instead offers an anonymous email option. Red properties are required *only** if you enable contact hiding:
  - `eaglei.repository.hideContacts` - true|false, enables the contact hiding function. When it is false, none of the other properties are used.
  - `eaglei.repository.postmaster` - email address of repository administrator(s). User-generated messages about resources without a contact email address get sent here, as well as diagnostic messages. We recommend using an email list or alias so it can be changed or directed to multiple people.
  - `eaglei.repository.mail.host` - hostname of SMTP server for outgoing mail, defaults to localhost.
  - `eaglei.repository.mail.port` - TCP port number of SMTP server for outgoing mail, only necessary if using a non-default port for your chosen type of service.
  - `eaglei.repository.mail.ssl` - Use SSL for connection to SMTP server for outgoing mail, value is true or false.
  - `eaglei.repository.mail.username` - Username with which to authenticate to SMTP server for outgoing mail, default is unauthenticated.
  - `eaglei.repository.mail.password` - password with which to authenticate to SMTP server for outgoing mail, default is none.

2.0 Release

The following **optional** properties are valid after the 2.0 release.

- `eaglei.repository.searchBar.javascript.url` - Location of the source of the JavaScript for the search bar. The default value is sufficient unless custom search bar code needs to be loaded.
- `eaglei.repository.centralSearch.url` - Location of the destination of the actual searches from the search bar. The default value is sufficient unless search is to be performed by a specific application.

Note that the properties file may also contain Log4J configuration properties. For example you can turn on debugging log output by adding this line:

```
log4j.logger.org.eaglei.repository=DEBUG, repository
```

Configuring Logging

The repository uses Apache log4j for its logging. Any properties starting with `log4j.` in the repository configuration properties are simply passed through to configure log4j. The Loggers (aka Categories) are all descendents of the **repository root Logger**, `org.eaglei.repository`, so you should configure the log level and appenders for that Logger.

Any log4j configuration properties in your repository configuration **replace** the corresponding defaults. Therefore, you **must** assign an appender for the repository root Logger, or it will not be able to log anything.

The default log4j configuration sets up an appender named repository with buffered I/O for efficiency. ***Note that this means log messages will not appear in the log file immediately, but only after the logging volume fills a buffer. This is useless for interactive debugging through the logs.*** If you are doing interactive debugging and want to see more log detail, along with immediate results, you should add the properties:

```
log4j.logger.org.eaglei.repository=DEBUG, repository
log4j.appender.repository.BufferedIO=false
log4j.appender.repository.ImmediateFlush=true
```

Also note that the default configuration turns off additivity in the repo root Logger; this means its log events do not propagate up to e.g. the root logger. If you wish to turn it back on, add this to your configuration:

```
log4j.additivity.org.eaglei.repository=true
```

Here are all of the default log4j configuration properties:

```
log4j.logger.org.eaglei.repository=INFO, repository
log4j.additivity.org.eaglei.repository=false
log4j.appender.repository=org.apache.log4j.RollingFileAppender
log4j.appender.repository.File=${eaglei.repository.log.dir}/repository.log
log4j.appender.repository.ImmediateFlush=false
log4j.appender.repository.BufferedIO=true
log4j.appender.repository.Append=true
log4j.appender.repository.Encoding=UTF-8
log4j.appender.repository.layout=org.apache.log4j.PatternLayout
log4j.appender.repository.layout.ConversionPattern=%d{ISO8601} %p %c - %m%n
```

IMPORTANT NOTE: If you add `logger` configurations to tweak the level of a subset of the repo log hierarchy, you **must** add an `additivity` configuration to prevent log4j from applying the ancestor logger as well, which would result in double log entries. For example, this fragment shows a default log level of INFO but adds DEBUG logging of RepositoryServlet to get elapsed time messages:

```
log4j.logger.org.eaglei.repository=INFO, repository
log4j.additivity.org.eaglei.repository=false
log4j.logger.org.eaglei.repository.servlet.RepositoryServlet=DEBUG, repository
log4j.additivity.org.eaglei.repository.servlet.RepositoryServlet=false
log4j.appender.repository.BufferedIO=false
log4j.appender.repository.ImmediateFlush=true
```

Monitoring and Troubleshooting

Version Information

It's often helpful to know exactly what version of the repository you're dealing with, especially in a hectic development and/or testing environment when many versions are available. The release version appears in these places:

1. Dissemination HTML pages, the **head** element contains a **meta** tag with the name `eaglei.version`, e.g.

   ```
   <meta name="eaglei.version" content="1.1-MS5.00-SNAPSHOT" />
   ```

2. The repository admin home page `/repository/admin` lists application version info in a human-readable format.
3. The page `/repository/version` gives a complete breakdown of component versions, including repo source and the version of the OpenRDF Sesame database. It is XHTML, and it includes **meta** tags to be easy to scrape or transform.

Log Files

Since the repository is mainly accessed by the REST service API it provides to other applications, you should get used to monitoring it by watching the log file. This is a text file (UTF-8 encoding) maintained by the log4j library under the control of the repository's configuration properties. See the description of the `log.dir` property above to learn the directory where logfiles are created; they are automatically rotated when the logfile grows too large.

The default repository logfile is in the `logs/` subdirectory of the repository home directory, and it is named `repository.log`. See the **Configuration Properties** section, above, for instructions on changing the destination directory for logfiles.

To troubleshoot problems with the logging system itself (e.g. log4j config that isn't working as expected), look for where your Java Servlet container writes the standard output stream. For Tomcat 6, this is typically the `catalina.out` file in some log directory.

Performance Monitoring

As of release 1.1MS5 the repo can log the elapsed time (in milliseconds) for each service request. You must enable DEBUG level logging for the RepositoryServlet, as in this configuration example.

```
log4j.logger.org.eaglei.repository=INFO, repository
log4j.additivity.org.eaglei.repository=false
log4j.logger.org.eaglei.repository.servlet.RepositoryServlet=DEBUG, repository
log4j.additivity.org.eaglei.repository.servlet.RepositoryServlet=false
log4j.appender.repository.BufferedIO=false
log4j.appender.repository.ImmediateFlush=true
```

As of release 1.2MS3 the repo will also show the time spent on internal SPARQL queries, which can be useful when tuning Sesame indexes. Add these log4j configuration lines to see ***just*** the query log messages:

```
log4j.logger.org.eaglei.repository.util.SPARQL = DEBUG, repository
log4j.additivity.org.eaglei.repository.util.SPARQL = false
```

Then, you'll see log entries like this which you can correlate to requests from your application:

```
...service invocation examples:

2011-01-27 14:28:06,483 T=http-8443-1 DEBUG org.eaglei.repository.servlet.RepositoryServlet -
```

```
============== Ending Request /repository/update (2,159 mSec elapsed)

2011-01-27 14:27:58,023 T=http-8443-1 DEBUG org.eaglei.repository.servlet.RepositoryServlet -
============== Ending Request /repository/workflow/push (1,763 mSec elapsed)

... (internal query example:)

2011-04-15 14:13:28,383 T=http-8443-1 DEBUG org.eaglei.repository.util.SPARQL -
SPARQL Query executed by
org.eaglei.repository.model.User:findAll at line 227 in elapsed time (mSec) 15
```

You can also get the SPARQL Protocol endpoint to make log entries at the INFO level for "slow" queries, i.e. ones that take longer than a certain threshold.

See the `eaglei.repository.slow.query` configuration property for more details. Note that this ***only*** applies to to queries made through the SPARQL Protocol endpoint, not the SPARQL queries generated internally by the repo code.

Tuning

The performance of Sesame's NativeStore implementation is extremely sensitive to its index configuration. There is a major benefit to configuring indexes that help resolve triple patterns used by the most frequent and/or voluminous SPARQL queries. A knowledgeable repository administrator should adjust the setting of the `eaglei.repository.sesame.indexes` property to get the NativeStore to build the most necessary indexes. See doc on that configuration for more details.

Administrator Tools

make-snapshot.sh Script

The make-snapshot script creates a ***complete backup copy*** of a data repository, in a designated directory. It has to be given a directory because the backup consists of multiple files. It is packaged with the repository distribution, under the `etc/` directory.

Upon success, the directory will contain two files:

1. `resources.trig` -- RDF resource data in TriG format, read by /graph
2. `users.trig` -- user accounts, must be read by /import serviceUpon failure, it prints an explanatory messaeg and returns non-0 status.

NO MESSAGE is printed upon success, which lets it run under cron.

Usage

Synopsis:

```
make-snapshot.sh username password repo-URL directory
```

Where:

- username - username with which to authenticate to the repo
- password - password with which to authenticate to the repo
- repo-URL - prefix of repository URL, e.g. "https://localhost/"
- directory - directory in which to write the dump, will be created if necessary

Restoring Dumps made by make-snapshot

Given a dump created in e.g. ${DUMPDIR}, to restore this dump on a newly-created, empty, repository, use these commands: (where ${REPOSITORY} is URL prefix of the repo)

```
curl -D - -s -S -u ADMIN:PASSWORD -F type=user -F format=application/x-trig \
-F content=@${DUMPDIR}/users.trig -F duplicate=replace \
-F transform=no ${REPOSITORY}/repository/import
```

```
curl -s -S -D - -u ADMIN:PASSWORD -F action=replace -F all= \
-F "content=@${DUMPDIR}/resources.trig;type=application/x-trig" \
${REPOSITORY}/repository/graph
```

Examples

For example, your crontab might invoke this command to write a daily snapshot

in a differently-named directory each day, rotating through a week:

```
make-snapshot.sh ADMIN PASSWORD https://localhost:8443 "daily_cron_`date +%u`"
```

move-everything.sh: Copying Everything Between Repositories or Files

The `move-everything.sh` script replicates ***all*** of a repository's contents - including resources, users and metadata - from one repository to a different one, or from a static file dump to a live repository. It ***transforms*** all resource (and user) URIs to match the URI prefix of the destination repository.

> **WARNING**
>
> This command obliterates all contents of the target repository.

Why do you need this script instead of just export and import requests? Because moving from one repo to another, the URIs of resources and users have to be transformed.

Since resource URIs have to be resolvable, this effectively creates new resources in the destination repository with URIs that resolve there. It does this by substituting the target's default prefix into all URIs that used to resolve at the source repository.

This Is Inherently Not A Good Idea

Before you start copying resources around, be sure you understand ***why this is not a good idea!*** Reasons include:

1. **Poor Semantic Web Hygiene**: Existing semantic web standards and technologies have ways to show that multiple URIs really describe the same object. This method does not use them, in the interest of making a precisely accurate copy of the data and not changing the source.
2. **Previous Content of Target Repo Is Lost**: We can't emphasize this enough because someone is sure to make a tragic mistake after not reading these instructions carefully enough. All contents of the repository you are moving resources into will be replaced by the copy of the source repository. ***All previous contents are lost irretrievably***. You made a backup, right?
3. **All State is Preserved**: All metadata for the state of e.g. data tools is copied faithfully, so claimed instances will have claims on the new site. This is not intuitive.
4. **User Accounts are added to Destination**: All of the user login accounts from the source are added to the target repo. Previously existing accounts are still available there too, but in the event of a duplicate, the target's account is replaced by the user URI and password from the source. ***You must be aware of any security issues this may create***.
5. **Abuse of auto-generated URIs**: Since the script is importing a bunch of URIs which were not generated by the native /new service, there is some chance of overlap. This should not happen so long as all repositories use the same time-based UUID generator for the suffix of URIs, but there is always a chance of conflicts.

However, move-everything has some advantages over move-resources:

1. **Copy is Close to Perfect**: The repository copy includes all self-contained resource instances, including unpublished resources and the user account instances referenced by administrative metadata.
2. **Blank Nodes included**: If your source data includes any blank nodes, they get copied too.
3. **Metadata should be complete**: The user account instances referenced by creator and contributor properties are copied as well so references will work.

Given all of these limitations, move-everything can still be an effective way of populating a repository for testing and demonstrations. Just stay aware of what doesn't work, and **only use it when the results are temporary and will be discarded**.

Restoring from Backups

There is one other legitimate use of **move-everything**: restoring a backup copy made with **make-snapshot**. In this case you don't really have to transform the URIs, and the whole intent is to re-create the original state of the repo so the side effects are all desired.

Using the Script

The resource copying script is installed under etc/ in the repository home directory. Its name is `move-everything.sh` . It only runs on a Unix-based operating system such as ***Linux*** or ***MacOS X***. It requires **bash**, **perl 5**, and the **curl** executable.

The synopsis for copying from **repository** to **repository**:

```
Usage: move-everything.sh [--version|--version] [ -f | --force ]
[-exclude-users user,user,..|-exclude-users user,user,..] [-nousers]
from-username from-password from-repo-URL
to-username to-password to-repo-URL
```

The synopsis for copying from **file** to **repository**:

```
Usage: move-everything.sh [--version|--version] [ -f | --force ]
[-exclude-users user,user,..|-exclude-users user,user,..] [-nousers]
--from-snapshot directory --from-prefix from-prefix
to-username to-password to-repo-URL
```

Options

The `--force` option: Normally the script starts up with a dialog explaining how dangerous it is and how the destination repo will be completely obliterated, and ask if you want to continue. Adding this option (abbreviated **-f)** will bypass the question and run every time, without asking. It is necessary when embedding it in another script. ***Only specify --force when you are very sure you're doing the right thing***. When prompted with the "Danger!" message, take time to actually ***read*** it before agreeing. You may be surprised.

If you specify a `--exclude-users` option, its value is a list of one or more usernames (separated by commas and/or spaces) to be left out of the source export. This is handy when you do not want to import the administrator accounts from the source system, for example. *Note that the excluded users' RDF metadata will **still** get copied, because it is in one of the named graphs which gets moved and transformed.*

If you specify the `--nousers` option (has no value), this turns off explicit copying of user accounts entirely. *Note that users' RDF metadata will **still** get copied, because it is in one of the named graphs which gets moved and transformed*. There will just be no login accounts. Also note this allows you to run move-everything without an Administrator login at the source repo, since all you need is read access to all the graphs - that does not necessarily require Administrator access.

The `--from-snapshot` and `--from-prefix` options must be specified together. They select the input data from a directory of serialized files, in the same format as produced by the make-snapshot script. The value of `--from-snapshot` is the path to the direcotry containing the RDF serialization files. The value of `-from-prefix` is the **exact and complete URI prefix** (including the trailing '*/*') of the repo that generated the dump in the directory. This is necessary because the script does not ahve access to that repository to query it for its prefix.

Fixed Arguments

The fixed command arguments are either one or two triplets of repository access information, i.e. the username, password, and URL of each repo.

If you selected file as input with `--from-snapshot`, then you must only specify the destination repository args. Otherwise, you specify first the **source** or **f rom** repository, and then the **target** or **destination** repo. Each set of args consists of:

- Username of the login account; must be Administrator on the target.
- Password for that login account.
- URL Prefix of the repository - just the part of the URL before "/repository/..."; e.g. `https://localhost:8443`

Here is an example that copies from the production Harvard repo to a local one:

```
move-everything.sh bigbird PASSWORD https://harvard.eagle-i.net \
bigbird PASSWORD https://localhost:8443
```

Here is an example that copies a snapshot the production Harvard repo to a local one:

```
make-snapshot bigbird PASSWORD https://harvard.eagle-i.net \
harvard.monday


move-everything.sh -f \
--from-snapshot harvard.monday \
--from-prefix http://harvard.eagle-i.net/i/ \
bigbird PASSWORD https://localhost:8443
```

Hints

We strongly recommend you **avoid** using the **Superuser** (administrator) login on the source repository, to prevent accidentally obliterating it by getting the argument order wrong. Use an account that has read access to every graph (e.g. the **Admin-Read-Only** role). This restricts you to using the --nousers version of the command but in most cases that is adequate. See the #Procedures section for recommendations on how to maintain copies of repositories this way.

move-resources.sh - Copying Only Resource Instances

The goal of this procedure is to copy all of the resource instances **in one Named Graph** from one repository to another, along with their relevant provenance and administrative metadata.

Since resource URIs have to be resolvable, this effectively creates new resources in the destination repository with URIs that resolve there. The hostname portion of the URI matches the new repository server, and even the local name is allocated by the destination repository -- so there is no predictable way to relate new URIs to the old ones.

This Is Inherently Not A Good Idea

Before you start copying resources around, be sure you understand **why this is not a good idea**! Reasons include:

- **Poor Semantic Web Hygiene**: Existing semantic web standards and technologies have ways to show that multiple URIs really describe the same object. This method does not use them, in the interest of making a precisely accurate copy of the data and not changing the source.
- **Copy is Imperfect**: The copy is bound to be missing some objects, and have some URIs that get translated without all of their descriptions being copied over. It's an inevitable consequence of the way data is stored in the repository. For example, the Person object of a dcterms:creator property may not be resolvable in the copy.
- **Blank Nodes ignored**: If your source data includes any blank nodes, their contents will probably not get copied or may get scrambled, since blank node identifiers are only unique within one site's repository.
- **Metadata gets Broken**: If you're copying from a file, the provenance metadata is all lost. If the source is a live repo, it gets copied, but the creator and contributor properties will refer to Person instances on the original repo. That's another reason "this is inherently not a good idea."

Given all of these limitations, the resource-mover script can still be an effective way of populating a repository for testing and demonstrations. Just stay aware of what *doesn't* work.

Using the Script

The resource copying script is installed under `etc/` in the repository home directory. Its name is `move-resources`. It only runs on a Unix-based operating system such as ***Linux*** or ***MacOS X***. It requires p**erl 5** and the **curl** executable.

Run it with -h to get the synopsis:

```
Usage: move-resources [-verbose] [-replace]
[--type published|workspace]{ --file source-file --prefix uri-prefix | --source source-repo-url
        --user login:password --graph src-graph-URI }
dest-repo-url dest-login:dest-password dest-graph-URI



(options may be abbreviated to first letter, e.g. -f)
```

By default it **adds** data to the destination graph, `--replace` changes that to replacing the entire graph.

You can change the type of the destination graph with the `--type` arg. E.g. set it to either **workspace** or **published**. By default the type is left alone.

You must choose a *source* by specifying either the *file* arguments (**-f** and **-p**), or *repository* arguments (**-s**, **-u**, **-g**). You must always specify the destination repository, login, and graph so they are plain args, not options.

Here is an example command, it copies from the **Published** graph on **qa.harvard** to an "Experimental" graph on the local repo (on `https://localhost:8443`)

```
move-resources -s https://qa.harvard.eagle-i.net:8443 -u bert:ernie \
-g http://eagle-i.org/ont/repo/1.0/NG_Published https://localhost:8443 \
root:password http://eagle-i.org/ont/repo/1.0/NG_Experimental
```

```
Moved 4694 data statements and 322 metadata statements.
```

Procedures

Procedure: Upgrading Packaged Tomcat

> **IMPORTANT**
>
> If you are using the Tomcat server from e.g. a Linux distro's package system, you must be aware of the following serious pitfall that can affect the repository when you upgrade Tomcat through the package system:

Some if not all packaged Tomcat servers include a sample webapp installed as teh ROOT webapp, so that the default server address can respond with a page congratulating you on installing Tomcat.

Meanwhile, the Repository ***replaces*** this ROOT webapp with its own (for good and compelling reasons detailed in the design documents). Thus, we destructively modify the installed state of Tomcat.

Some Tomcat package upgrade proceduers (notably Fedora Core 12) have been observed to simply replace files in the expanded ROOT webapp without checking that it was the original default ROOT webapp installed from the package. While we consider this a serious bug in the distribution package, it is unlikely to be fixed, so you must learn to expect and recover from it.

So, **after** upgrading a packaged Tomcat:

Remove the ROOT webapp directory (with Tomcat still shutdown) and the `${CATALINA_HOME}/webapps/ROOT.war` file, to ensure any replaced or corrupted version is gone.Replace it with the ROOT.war from your Repository distribution - you DID save it, right?

Finally, delete the entire `${CATALINA_HOME}/work` direcotry. Tomcat rebuilds it on startup anyway, but it can contain mistaken caches that do not get updated. Now you can start up Tomcat as usual.

Procedure: Installing Repo on Ubuntu 10's packaged Tomcat6

See also: The **Procedure to redirect Port 80** so your URLs are simplified.

This section describes the differences in the install procedure when using the packaged tomcat6 server on Ubuntu Linux 9.10 (karmic koala). It was based on experience with package tomcat6, version 6.0.20-2ubuntu2.1 .

NOTE: This procedure only lists the steps specific to Ubuntu's tomcat package. You need to review the previous section and follow that procedure, referring to this one for the steps related to tomcat.

1. **Shut down tomcat**. This is major surgery, and tomcats don't like to be vivisected no matter how much more satisfying you may find it.
2. **Disable Java Security** -- alternately, you could try to configure all the authorization grants to give the repository webapp access to the filesystem and property resources it needs, but I found it much easier to just disable java security. ***DO NOT RUN THE TOMCAT PROCESS AS ROOT*** if you do this, but you should not be running it as root in any case. That's just insane.
    a. Edit the file `/etc/init.d/tomcat6` and change the following variable to look like this:

    ```
    TOMCAT6_SECURITY=no
    ```

3. **Install Derby jars**: ONLY IF DERBY IS NOT ALREADY INSTALLED IN THE COMMON AREA OF YOUR TOMCAT. If another webapp is already using Derby, they should share that version.
    a. Find the Derby jars in the `lib/` subdirectory under where you installed the `create-user.sh` script.
    b. Copy them to the Tomcat common library directory:

    ```
    cp ${REPO-ZIP-DIR}/lib/derby* /usr/share/tomcat6/lib/
    ```

4. **Install the webapp**: First, get rid of any existing root webapp, then copy in the webapp (`ROOT.war` file from your installation kit) and be sure it is readable by the tomcat6 user:

    ```
    rm /var/lib/tomcat6/webapps/ROOT*cp ROOT.war /var/lib/tomcat6/webapps/ROOT.war
    ```

5. **Install cached webapp context**: This is *VERY IMPORTANT*, and the Tomcat docs does not even mention it, but without it your server will be mysteriously broken. The file `/etc/tomcat6/Catalina/localhost/ROOT.xml` must be a copy of your app's `context.xml`. Redo this command after installing every new `ROOT.war`:

```
mkdir -p /etc/tomcat6/Catalina/localhost
unzip -p /var/lib/tomcat6/webapps/ROOT.war META-INF/context.xml > /etc/tomcat6/Catalina/localhost/ROOT.
xml
```

6. **Add System Properties**: Be sure you have added system properties to the file `/etc/tomcat6/catalina.properties` e.g.

```
org.eaglei.repository.home = /opt/eaglei/repoderby.system.home = /opt/eaglei/repo
```

...of course, the value of these properties will be your Repository Home Directory path.

7. **Start up Tomcat**:

```
sudo /etc/init.d/tomcat6 start
```

8. **Troubleshooting**: If there are problems, check the following places for logs (because packaged apps make everything so much easier):
    - `/var/log/daemon.log` - really dire tomcat problems and stdout/stderr go to syslog
    - `/var/log/tomcat6/*` - normal catalina logging
    - `${REPOSITORY_HOME}/logs/repository.log` - default repo log file in release 1.1; under 1.0 the filename was `default.log`.

Procedure: Run Tomcat on Port 80 (and 443)

We want the repository (and other Web tools) to have a simple URL, without the ugly port number after the hostname, e.g. like this http://dev.harvard.eagle-i.net/ and NOT http://dev.harvard.eagle-i.net:8080/`...` (because really, that first one is already enough to remember.) This procedure uses IP port redirection to let your Tomcat server appear to be running on the canonical HTTP port, which is 80. It is the simplest and safest method to accomplish this under Linux.

The sanest alternative, running an Apache httpd server as an AJP forwarder, is much more effort and adds another point of failure. We will not even discuss running Tomcat as root so it has access to port 80, since that is simply unacceptable.

These procedures

- have been tested under Ubuntu Linux 9.10 _(krazy kitten), Fedora 12 and 14, and CentOS 6.03
- assume you are running Tomcat on port 8080. To redirect the HTTPS (HTTP on SSL) port, also run the 3 additional **iptables** commands (assuming port 443) below.
- require root privileges
- assume the Bourne shell (/bin/sh)

1. To check the what rules are running

```
iptables -t nat -n -L
```

2. Discover your machine's primary IP address and set the ADDR shell variable: (Note that this assumes **eth0** is your primary network interface -- use `ifconfig -a` to see them all)

```
ADDR=`ifconfig eth0 | perl -ne 'print "$1\n" if m/\sinet addr\:(\d+\.\d+\.\d+\.\d+)\s/;'`
```

3. Run these iptables commands to redirect all port 80 requests to port 8080.

```
iptables -t nat -A OUTPUT -d localhost -p tcp --dport 80 -j REDIRECT --to-ports 8080
iptables -t nat -A OUTPUT -d $ADDR -p tcp --dport 80 -j REDIRECT --to-ports 8080
iptables -t nat -A PREROUTING -d $ADDR -p tcp --dport 80 -j REDIRECT --to-ports 8080
```

4. (If using SSL) Run these iptables commands to redirect all port 443 requests to port 8443.

```
iptables -t nat -A OUTPUT -d localhost -p tcp --dport 443 -j REDIRECT --to-ports 8443
iptables -t nat -A OUTPUT -d $ADDR -p tcp --dport 443 -j REDIRECT --to-ports 8443
iptables -t nat -A PREROUTING -d $ADDR -p tcp --dport 443 -j REDIRECT --to-ports 8443
```

5. Check that your new rules are running (use the command above)
6. Additional configuration
    a. Ubuntu
        i. Save the rules in the canonical place to be reloaded on boot:

```
iptables-save > /etc/iptables.rules
```

ii. Create a script to be run by the network startup infrastructure that will reload the iptables whenever the network is configured on:

```
cat << EOF > /etc/network/if-pre-up.d/iptablesload
#!/bin/sh
iptables-restore < /etc/iptables.rules
exit 0
EOF
```

b. Fedora

i. Save the rules to be reloaded on boot:

1. The cleaner/preferable method, but apparently **not** working:

```
/sbin/iptables-save
```

2. Hacky, but works: manually edit `/etc/sysconfig/iptables`

ii. Update the startup settings so iptables will run upon reboot:

```
chkconfig --level 35 iptables on
```

7. Test by accessing your server both locally and remotely by the port-80 URL. Then reboot the machine and try it again to be sure the iptables commands are run correctly on boot.

Procedure: Dump and Restore the RDF Resource Data

The recommended way to dump out the RDF resource data content of the repository is to *export* it as *serialized RDF*. If you are exporting the entire contents of the repository, it is essential to preserve the mapping of statements to named graphs, so you must use one of the formats that encodes RDF as quads (statement plus graph-name/context).

The reason for this is that the repository server employs an RDF database, Sesame, to manage the RDF statements. It uses Sesame's "native" store, which records statements in opaque data files on the host OS's filesystem -- but much like relational database systems, Sesame's files are never in a consistent state while it is running so it would have to be shut down (by shutting down the repository Web service) to make a "cold" snapshot backup. It is much easier to simply export the live data. Another advantage of exports as backups is that the data can easily be *imported* into a later version of Sesame or even a different database.

This is a complex **manual** procedure with many options -- for a simpler semi-automated backup snapshot procedure, see the section on using the **make-snapshot** script.

Make Backup Dump (obsolete - see make-snapshot)

Typical command to make a backup, in TriG format to a file, e.g. all-dump.trig (here highlighted in yellow) from a server running locally on port 80. In practice, you'll probably need to change all the highlighted parts, such as the `username:password` login credentials, and the hostname in the target URL if not running locally.

```
curl -G -X GET -s -S -u username:password -o all-dump.trig -d all \
--write-out 'status=%{http_code}, %{time_total}sec\n' \
-d format=application/x-trig https://localhost:8443/repository/graph
```

Be **sure** the output shows a successful status code (namely **200**), as shown here, since curl will return a successful status even if the HTTP service did not succeed; curl only reports on the success of the network request-and-response transaction.

```
status=200, 13.283sec
```

Restore Repository from Backup

NOTE: This form of the procedure is a bit obsolete, since the new move-everything.sh script can also restore the state of a repository from its own backup -- effectively moving data to itself. See that command for details.

The procedure is still worth mentioning since it demonstrates the nature of the backup's contents:

Typical restore command.

WARNING

this replaces the entire contents of the repository!

```
curl -s -S -u username:password -F action=replace -F all= \
--write-out 'status=%{http_code}, %{time_total}sec\n' \
-F 'content=@all-dump.trig;type=application/x-trig' https://localhost:8443/repository/graph
```

Be **sure** the output shows a successful status code (namely **201**, since it created graphs), as shown here, since curl will return a successful status even if the HTTP service did not succeed; curl only reports on the success of the network request-and-response transaction.

```
status=201, 13.283sec
```

Procedure: Saving and Restoring User Accounts

As of the MS6 release, you can use the new Export/Import service to create user accounts automatically (e.g. on a newly-created repository). This is *NOT* the same thing as true backup and restore; rather, it is intended more for setting up a test environment. The export and import services are very complex and powerful. This only gives one small example of what they can do. For all the details, see their **entry in the API Manual**.

Step 0. Create Prototype Accounts and Export Them

Only do this once. Once you create a user file you like, you can use it over and over, on any different sites and tiers you like.

Create the user accounts you want on some repository instance. You will export them to create a document describing the user accounts you want. There can be extra accounts, you can filter them out of the export. So, get all the accounts you want in order, with roles, passwords, and personal names set up.

Now run a command like this to export the accounts into the file `all-users.trig`

```
curl -s -S -u username:password -G -d type=user -d format=application/x-trig \
--write-out 'status=%{http_code}\n' \
-o all-users.trig https://hostname:8443/repository/export
```

Note that you have to change the **hostname** and possibly the login. If there are accounts you do not want in the export, add an **exclude** argument to filter them out, with a space-separated list, e.g.

```
.... -d 'exclude=frankenstein moreau lizardo' ....
```

Step 1. Import Accounts on Destination Sites

You can start with a newly-created repository which needs to have user accounts added. It only has the initial administrator login, e.g. `bigbird`. Use the import service to add users from the file you created in step 0. The following command adds all of the accounts *except* **bigbird** (since it already exists), and aborts without changing anything if there are already duplicates of any of the users on the destination repo. It will print "status=200" on success.

```
curl -s -S -u username:password -F type=user -F format=application/x-trig \
-F transform=yes --write-out 'status=%{http_code}\n' \
-F exclude=bigbird \
-F content=@all-users.trig https://hostname:8443/repository/import
```

Note that the **transform=yes** argument means *import* will translate the instance URIs of the new users to newly-created URIs in the repository's default namespace. This is usually what you want. If you are positively restoring users already in the correct namespace and you want to preserve the old URIs, substitute **transform=no**.

Step 2. Testing Users

The easiest way to test the existence and details of a user is with the **/whoami** service. It does not show roles, however, you'll have to go to the repository administrative UI for that (or take it on faith). For example, after restoring users including **curator**, this is how you'd check that curator exists:

```
curl -s -S -u curator:password -G -d format=text/plain https://hostname:8443/repository/whoami
```

It's probably only necessary to test one user like this, and to make sure the output includes a URI, as a check that the whole import succeeded.

Procedure: Exporting and Importing Property Access Controls

This is only relevant to release 1.5MS1 and later, when resource properties have access controls.

To determine the URIs of the access controls, bring up the admin UI pages and login as an Administrator. There will be a link to the Properites access control page named Manage Property Access Controls. If you go to that page, it will display two sets of properties for which there is an access control list:

1. "Hidden" properties
2. Contact properties

Go to each of these in turn and observe the URI of the subject, e.g. `http://eagle-i.org/ont/app/1.0/PropertyGroup_AdminData`. This is the URI to *include* in your export request. Now do the same for contact properties and record that URI too.

To export property grants, plug those URIs into the following command (you need to replace italicized words):

```
curl -G -k -u ADMIN:PASSWORD -d type=grant -d "include=HIDE,CONTACT" \
-d format=application/x-trig https://localhost:8443/repository/export
```

This writes a record of grants to the standard output. Since the URIs are the same between other repositories running the same data model, you should be able to import them with the command (shows standard input in the example):

```
curl -k -u ADMIN:PASSWORD -F type=grant \
-F duplicate=abort -F transform=no -F content=@- \
-F format=application/x-trig https://localhost:8443/repository/import
```