Repository Data Model Configuration Properties

Contents

This page describes the way a *data model ontology* is configured into the repository. This ontology is like (but not really equivalent to) a schema for RDF data, it predicts what you are likely to find in the data and describes how you should react to it. For example, the ontology describes which properties of a resource instance should not be exposed to the public, and this configuration tells the repository where to look in the ontology for those rules.

Introduction

The repository is designed to be independent of any data model ontology. All of its interactions with the ontology are externally configured, through a properties file. This should allow other projects to use the eagle-i repository as a standalone component, incorporating their own data model.

The data model configuration is contained in a properties file *separate* from the main repository configuration. Since the data model configuration is:

1. uniform among all repository instances for an application
2. unlikely to be changed
3. contains very complex and specialized information

It makes more sense to keep it separate from an instance's configuration, reserving that for instance-specific variables.

Locating the Configuration

The data model configuration file is expected to be packaged as a resource in the repository's web application (webapp). Since it is so crucial to the operation of the repository, this ensures it is always available and is not subject to an administrator's mistakes.

To allow alternate data model configurations to be selected, there is a configuration property in the main repository configuration which names the resource path of the data model configuration: **eaglei.repository.datamodel.source**. Its default value is the path "eaglei-datamodel.properties", which describes the eagle-i data model. You do not need this property in the system configuration if this default is acceptable.

Format

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

You *can* treat it simply as a Java properties file, but be aware that the comma (,) character has special significance in property values. It is used to specify a list of values, and if the value is ***not*** expected to be a list, everything after the first comma is ignored.

About Versions

Although your data model ontology may not contain any RDF statements at all, it is ***required*** to have at least one statement describing its *version*, so the loaded version can be compared with the one available in the webapp when considering an update and generating reports. (See the **/model** service in the REST API for details.)
The version statement consists of an arbitrary URI as subject, the predicate http://www.w3.org/2002/07/owl#versionInfo, and a literal string value that is expected to start with a Maven-style version number (a sequence of integers separated by dot (.). For example:

```
<http://purl.org/eagle-i/app-ext/> <http://www.w3.org/2002/07/owl#versionInfo> "0.7.6 8274  2011-03-28 15:06:29"
```

As you will see in the next section, the version is also key to describing the location of the data model as well.

Loading the Data Model

See the description of the property

Unknown macro: {datamodel.versionInfo.source}

for the gory details of how the repository actually loads the data model. Although it seems somewhat convoluted, it lets you avoid specifying redundant information, such as the JAR file containing the ontology, since it can be derived from what was already given by making some safe assumptions.

Properties in the Data Model Configuration

Ontology Metadata

These properties describe the data model ontology itself, and how it is managed by the repository.

- **datamodel.graph.uri**
  The value is the URI naming the named graph were the data model ontology statements are to be stored. The repository adds statements to this graph when it loads the data model.
- **datamodel.graph.label**
  The human-readable label to be set on the data model's named graph. This is purely decorative and will be seen in the Admin GUI and **listGraphs** service.
- **datamodel.versionInfo.source**
  This property names the file in the data model containing the relevant `owl:versionInfo statement`. It has two distinct purposes:
  - First, indicate the resource file containing the statement describing the actual released version of this data model ontology.
  - Second, as a fortunate side-effect, identify the container or archive that can be expected to contain the data model ontology files. The current algorithm for loading the data model ontology is to locate this container (which is assumed to be a JAR archive file) and *load every file in it a name ending in* ".owl". This is a crude assumption that works for ontologies maintained in Protoge.

- **datamodel.versionInfo.uri**
  Identifies the subject URI of the statement expected to contain the version of the data model ontology. The predicate is, as has been noted, `owl:versionInfo` (see section on Versions above).

Data Model Description

These properties direct where the repository looks in the ontology to control some of its actions, e.g. to decide what properties of a resource instance are to be hidden from public view.

Label Properties

When the repository prepares a human-readable description of a resource instance, it attempts to display the text *label* for all URIs in the subject, predicate, and object of the instance's graph. Also, when computing the graph to return for an RDF dissemination request, it includes the labels of all of these objects.

Since some ontologies use other properties for labels instead of or in addition to the conventional `rdfs:label`, the repository can be configured with an ordered list of label properties. Its influence is as follows:

1. In a human-readable dissemination of the resource instance, the *first* label predicate with a value is shown in place of the URI. If none have values, then the URIs local name is displayed.
2. In a dissemination of serialized RDF, values for *any and all of the label predicates* are included, in no particular order.

The configuration property is:

- **datamodel.label.predicate**
  Value is a comma-and-or-space separated list of URIs to be used as label predicates. For example:

```
datamodel.label.predicate = http://eagle-i.org/ont/app/1.0/preferredLabel, \
http://purl.obolibrary.org/obo/IAO_0000111, \
http://www.w3.org/2000/01/rdf-schema#label
```

Embedded Instances

- **datamodel.embedded.predicate**
  This defines the *predicate* of a statement *in the data model ontology* whose subject URI is identified as the name of an embedded instance class.
- **datamodel.embedded.object**
  This defines the *object* of a statement *in the data model ontology* whose subject URI is identified as the name of an embedded instance class.

Any class URIs in the ontology which are the subject of statements with the given predicate and object are considered *embedded object classes*. This is significant in the implementation of the repository and will not be covered here. For example, here is the configuration, and a statement in the ontology describing an affected class:

```
datamodel.embedded.predicate = http://eagle-i.org/ont/app/1.0/inClassGroup
datamodel.embedded.object = http://eagle-i.org/ont/app/1.0/ClassGroup_embedded_class
```

```
# some embedded instance classes
@prefix ero: <http://eagle-i.org/ont/app/1.0/> .
<http://purl.obolibrary.org/obo/OBI_1110023> ero:inClassGroup ero:ClassGroup_embedded_class .
<http://purl.obolibrary.org/obo/ERO_0000404> ero:inClassGroup ero:ClassGroup_embedded_class . ...
```

Hidden Properties

These properties tell the repository how to identify properties that should be hidden from public view. The exact rules about how they are to be exposed are not designed yet.

In practice, the predicate and object configuration lines work exactly the same as for embedded instances above, only they name the URIs of *properties* that are to be hidden.

- **datamodel.hideProperty.predicate**
  This defines the *predicate* of a statement *in the data model ontology* whose subject URI is identified as the name of a *hidden property*.
- **datamodel.hideProperty.object**
  This defines the *object* of a statement *in the data model ontology in the data model ontology* whose subject URI is identified as the name of a *hidd en property*.

Contact Properties

These configuration keys define how the repository detects *contact properties*, i.e. properties that contain information about real-world contacts such as e-mail addresses, phone numbers, lat/long, etc. There are some situations where this information must be kept confidential, so this is how it is identified. The exact rules about how they are to be exposed are not designed yet.

The predicate and object configuration lines work exactly the same as for hidden properties above.

- **datamodel.contactProperty.predicate**
  This defines the *predicate* of a statement *in the data model ontology* whose subject URI is identified as the name of a *contact property*.
- **datamodel.contactProperty.object**
  This defines the *object* of a statement *in the data model ontology in the data model ontology* whose subject URI is identified as the name of a *cont act property*.

Contact Email Address

These configuration properties describe how the repository locates a *contact email address* for a given resource instance. This address is where email generated by the "contact the owner" form (see the Repository API for details).

- **datamodel.contact.email.query**
  The text of the SPARQL query to find the email addresses. It is expected to be a SELECT query that binds the variables mentioned in the bindings list below. You can use a backslash at the end of the line to escape newlines in the query.
- **datamodel.contact.email.bindings**
  An ordered list of the query variables that can be bound by the email query. Each variable is identified only by its name, *without* the leading "?". They are separated by comma and/or space. The *first* one of these with a value is used as the email address. If there are multiple results to the query, any additional values of ***that*** variable (and no others) are added as destination addresses.

Here is an example:

```
datamodel.contact.email.bindings = email1,email2,email3, email4
datamodel.contact.email.query = \
SELECT ?email1 ?email2 ?email3 ?email4 WHERE { \
OPTIONAL { \
?subject <http://purl.obolibrary.org/obo/ERO_0000021> ?person . \
?person <http://purl.obolibrary.org/obo/ERO_0000041> ?email1 . } \OPTIONAL { \
?subject ?any_predicate ?organization . \
?any_predicate <http://eagle-i.org/ont/app/1.0/inPropertyGroup>  \
<http://eagle-i.org/ont/app/1.0/PropertyGroup_RelatedResourceProvider> . \
OPTIONAL { \
?organization <http://purl.obolibrary.org/obo/ERO_0000021>
?person . \       ?person <http://purl.obolibrary.org/obo/ERO_0000041> ?email2 . \
filter(!bound(?email1)) } \     OPTIONAL { \
?organization <http://purl.obolibrary.org/obo/ERO_0000041> ?email3 . \
filter(!bound(?email1) && !bound(?email2)) } \
OPTIONAL { \
?organization <http://purl.obolibrary.org/obo/ERO_0000022>
?pi . \       ?pi <http://purl.obolibrary.org/obo/ERO_0000041> ?email4 . \
filter(!bound(?email1) && !bound(?email2) && !bound(?email3)) \     }}}
```