Code-Documentation

Code – Documentation

Operational Use

Prerequisites

JDK 1.6 http://www.oracle.com/technetwork/java/javase/downloads/index.html

Maven 2 http://maven.apache.org/

Tomcat http://tomcat.apache.org/

NetBeans IDE (recommended) http://netbeans.org/

Provide Persistence

Data persistence is provided using a MySQL relational database.

Initial Creation of vfrbr Database

Creation of the database presumes an existing installation of MySQL, with an established user and password.
The initial installation of MySQL results in the default creation of a "root" user with no password.
With this default condition, the **`mysql`** client can be accessed thusly:

```
system-prompt$ mysql --user="root" --password=""
```

Note that an empty quoted parameter is required for no password.

An initial database space can be created with the mysql client command:

```
mysql> create database vfrbr
```

This creates a database named "vfrbr". There is nothing special about using "vfrbr" for the database name. Whatever name used will be specified

in subsequent program settings.

The tables for a vfrbr persistence database can be created with SQL files contained in the **`vfrbr-persist`** project.

```
vfrbr-persist/
  src/
    main/
      resources/
        sql/
          ddl/
            create-constraints.sql
            create-tables.sql
            drop-constraints.sql
            drop-tables.sql
            fiz/
              create-fiz-tables.sql
              drop-fiz-tables.sql
```

To create the vfrbr tables, execute the following SQL files from the source directory, or with path prepended to the file names. Once the database has been created, the mysql client can be invoked with the database as a parameter.

```
system-prompt$ mysql --user"root" --password"" vfrbr
mysql> source create-tables.sql
mysql> source create-constraints.sql
mysql> source fiz/create-fiz-tables.sql
```

Emptying vfrbr Database for New Load

If a new frbrization load is to be done subsequent to a previous one, the database tables should be emptied. This can be done by sequentially dropping the constraints, dropping the tables, and creating the tables and constraints by execution of the following sequence of SQL files.

```
mysql> source drop-constraints.sql
mysql> source drop-tables.sql
mysql> source create-tables.sql
mysql> source create-constraints.sql
mysql> source fiz/drop-fiz-tables.sql
mysql> source fiz/create-fiz-tables.sql
```

Dropping the constraints on a fully loaded database can take some several minutes. But the other files all execute very quickly.

vfrbr-persist Maven Project for Java Persistence

The **`vfrbr-persist`** maven Java project provides the linkage between Java and the relational database. The project contains a project of Java classes that represent an implementation of an extended FRBR model. The Java classes have annotations that map from Java to the relational database tables, as provided by the Java Persistence API (JPA).

The project needs to be compiled and build through maven to be installed into a maven repository to be available as a dependency for the other program projects providing FRBRization loading, data exporting, and Scherzo user search interface.

A vfrbr database needs to exist before compiling. The user, password, and JDBC connection URL are specified in the **`jdbc.properties`** file. The JPA settings are specified in the **`persistence.xml`** file.

```
vfrbr-persist/
  src/
    main/
      resources/
        jdbc.properties
        META-INF/
          persistence.xml
```

Top

FRBRize MARC Data

The FRBRization process reads MARC bibliographic record files, and using MARC authority record files, converts the bibliographic data into an extended FRBR form persisted into relational database tables.

The program code to perform FRBRization is contained in the **`vfrbr-frbrize-marc`** maven Java project.

Dependencies of FRBRization

The **`vfrbr-frbrize-marc`** project has a maven dependency upon the **`vfrbr-persist`** project. For the frbrization project to compile the persistence project needs to have been compiled and installed into an available maven repository. The vfrbr relational database is used by the persistence project in supporting the frbrization project.

Setup for FRBRization

vfrbr Database

The database that will be used is determined by the **`vfrbr-persist jdbc.properties`** properties file.
The database should be empty unless an incremental loading is being performed.

MARC Authority Record Files

The frbrization process allows for the use of authority files from a service like OCLC's. These files are accessed via Z39.50 and are cached locally on the file system. The locations of the files are specified in the **`vfrbr-frbrize-marc authCache.properties`** properties files. There are separate directories for work, people, and corporations.The properties file is also where you input information about the authority service you are using as well as login information. If you do have access to this delete the file AuthorityHandler.java and comment all references to it (this is most easily done in an IDE such as NetBeans or Eclipse). If you do want to make use of authority files you must install local Z39.50 client development software. We use YAZ for this -- https://www.indexdata.com/yaz. In order to use YAZ in Java code you also need to install SWIG (http://www.swig.org/) and g++ (http://directory.fsf.org/project/gpp/). Once all that has been installed download YAZ4J (http://www.indexdata.com/yaz4j) and use maven to install it to you local repository by simply typing mvn install in the YAZ4J directory. The maven script will inform you of any problems. Once fetched the frbrization application will use these cached files. If you want to fetch new versions of these files simply delete the directory you set in the authCache.properties file.

MARC Bibliographic Record Files

The frbrization processes MARC bibliographic record files.
The location of the files is specified in the **`vfrbr-frbrize-marc batchLoading.properties`** properties file. The names of the files are hard coded in the **`BatchLoading.java`** program.

Project Settings for FRBRization

The property files (vfrbr-frbrize-marc/src/main/resources/):

- **`authCache.properties`**:
  should hold the directory paths for the cached MARC authority files;
- **`batchLoading.properties`**:
  should hold the directory path for the MARC data files,
  the **`persist_unit_name`** property is not used;
- **`compositionFormCodes.properties`**:
  holds the codes and values for composition forms;
- **`countryCodes.properties`**:
  holds the codes and values for country names;
- **`jdbc.properties`**:
  is not used for FRBRization
  (but is used for generating analysis report spreadsheet data);
- **`languageCodes.properties`**:
  holds the codes and values for language names;
- **`log4j.properties`**:
  holds the settings for logging files.

The MARC bibliographic record files to process are coded in the BatchLoading.java program.

BatchLoading.java

```
...
    private final String[] MARC_FILES = {
        "frbr01.mrc",
        "frbr02.mrc",
        "frbr03.mrc",
        "frbr04.mrc",
        "frbr05.mrc",
        "frbr06.mrc",
        "frbr07.mrc",
        "frbr08.mrc",
        "frbr09.mrc",
        "frbr10.mrc",
        "frbr11.mrc",
        "frbr12.mrc",
        "frbr13.mrc",
        "frbr14.mrc",
        "frbr15.mrc",
        "frbr16.mrc",
        "frbr17.mrc",
        "frbr18.mrc",
        "frbr19.mrc",
        "frbr20.mrc"
    };
...
```

Selective commenting of the source code can limit the processing to a subset of the files.

Execution of FRBRization

The FRBRization process uses a rather large amount of program memory space to execute.
Successful execution of the program has been using options to run the program as a separately forked process with a VM parameter of **`-Xmx3500m`** to ensure enough execution space.
Running locally on a MacBook Pro with 2.4 GHz Intel Core 2 Duo processor, with a local MySQL database, the FRBRization of all twenty files takes about six and a half hours (400 min.).
To capture a log of just one FRBRization run, ensure that any existing log file at the log4j properties destination setting has been deleted.

As a Maven-based project the **`vfrbr-frbrize-marc`** Java program can be executed from within an IDE context that supports integration with Maven or from a command-line invocation through the **`mvn`** command. To execute via maven type

```
mvn exec:java -Dexec.mainClass="edu.indiana.dlib.vfrbr.frbrize.batchloading.BatchLoading"
```

Products from FRBRization

After the FRBRization run, the MySQL database will be loaded and a processing log file will be created.
The log file reports all the MARC records processed and the details of the processing. The file will be quite large (1.27 G) but can be quite useful for analyzing the details of FRBRization processing. The **`less`** viewing utility has worked well for examining the log file, since it does not need to read the entire file into a buffer.
The other useful product to produce is a portable dump of the database, created with the MySQL **`mysqldump`** utility program. This produces an output of SQL commands that can be used as input to create a duplicate database. Redirecting the output to a file (928 MB) records the output into a backup record of the database.

```
system-prompt$ mysqldump --user"root" --password"" vfrbr > mysqldump.sql
```

The backup file can be imported into an existing database using the input redirection feature of the mysql client. This would typically be done with an empty vfrbr database.

```
system-prompt$ mysql --user"root" --password"" vfrbr < mysqldump.sql
```

Note that the use of the name "mysqldump" for the sql backup file is only convention.

Top

Export FRBRized Data

FRBRized data can be exported from the persistence database in the form of XML based on the 1.0 or 1.1 Schema namespace files, or RDF/XML with OWL namespace declarations structured similiar to the 1.1 XML Schema structure.

Dependencies of Exporting

The **`vfrbr-export`** maven project depends upon the **`vfrbr-persist`** project being available from a maven repository and having a loaded vfrbr database as specified in the **`jdbc.properties`** properties file of the **`vfrbr-persist`** project.

Setup for Exporting

The determination of which elements to export is established by inclusion or commenting of lines within the **`XMLExport.java`** main program.

For example, the following code would export all elements:

Exporting.java

```
...
    public void exportingTest() {
        exportWorks();
        exportExpressions();
        exportManifestations();
        exportPersons();
        exportCorporateBodies();
        exportStructRealized();
        exportStructEmbodied();
        exportResponCreated();
        exportResponRealized();
        exportResponProduced();
    }
...
```

and this code would just export Works:

Exporting.java

```
...
    public void exportingTest() {
        exportWorks();
//        exportExpressions();
//        exportManifestations();
//        exportPersons();
//        exportCorporateBodies();
//        exportStructRealized();
//        exportStructEmbodied();
//        exportResponCreated();
//        exportResponRealized();
//        exportResponProduced();
}
...
```

The determination of which kind of export to produce is established by inclusion or commenting of lines within the **`Export.java`** context .

For example, this setting is for an XML 1.1 export of efrbr.

ExportContext.java

```
...
     /*
     *  Default initializations.
     */
    static {
        uriStem = "http://vfrbr.info/";

//        fileDirPath = "/usr/local/vfrbr/exports/xml/1.1/";
fileDirPath = "/usr/local/vfrbr/exports/rdf/";

        scheme = "XML";

        version = "1.1";

        level = "efrbr";
    }
...
```

The supported combinations for efrbr export types are:
*XML 1.0
*XML 1.1

For each element, there is a context file that specifies the stem of the export file name, the number of records to include in a file (multiple files have serial numbering in the file name), whether only a sample of data elements is being exported, and for a sample the beginning and ending elements being sampled.

For example, the context file for Works:

WorkContext.java

```
...
    public WorkContext() {

        fileStem = "work";

        recsPerFile = 5000;

        sampleExport = false;

        sampleFromIndex = 0;

        sampleToIndex = 10;
...
```

The same configuration for RDF can be found in the class **BaseEntityExporter.java** at the top of the file. The RDF exporter is set to always export both frbr and efrbr levels.

Execution of Exporting

The exporting program can be executed in the same manner as the **`BatchLoading.java`** program, and should include the VM parameter for maximum memory size and be executed as a separately forked process. Run XMLExport or RDFExport depending on what you wish to export.

Top

End User Search Interface (Scherzo)

The vfrbr-search download contains two separate Maven projects. One called vfrbr and one called vfrbr-search. The vfrbr-search project interacts with a database of FRBRized records and an embedded version of Solr (a popular search server) to build the search indexes. The vfrbr project contains the user interface and depends on vfrbr-search.

Building the index

The vfrbr-search project expects that you have already set up your vfrbr-persist project and database and have loaded it with records via the vfrbr-frbrize-marc project as documented above. If you have your vfrbr-persist project set up correctly there is only line that needs changed to configure vfrbr-search. That line is located in /vfrbr-search/src/main/resources/solrconfig.xml on line 39.

solrconfig.xml line 39

```
<dataDir>/usr/local/vfrbr/solr</dataDir>
```

Simply change this to a directory that tomcat can read from and is where you want your indexes to be stored.

Now all you have to do is run the class edu.indiana.dlib.vfrbr.search.solr.Indexer. Again this can be done in an IDE or can be done with maven on the command line. First enter the base directory for vfrbr-search (it will have the file pom.xml in it) and type:

```
mvn install
mvn exec:java -Dexec.mainClass="edu.indiana.dlib.vfrbr.search.solr.Indexer"
```

This can take some time to run, but you should now have a search index built and are ready to run the end user search interface against it.

Working with Scherzo

Working with Scherzo requires some familiarity with the Tomcat Application Server. The official Tomcat website (http://tomcat.apache.org/) is a great resource for learning how to use Tomcat. The search application itself gets all of its configuration from its dependencies so all you need to do to run Scherzo is build it with maven and deploy the "war" file that maven produces. You build this application the same as the rest by moving to the vfrbr directory in console and typing "mvn install" or building in your IDE. After building there will be a new subdirectory under vfrbr called "target" and this directory contains the file scherzo.war. Deploy this file to your Tomcat instance on the same machine that has the indexes on, it and visit the URL of you application (something like http://localhost:8080/scherzo or http://tomcat.university.edu:8080/scherzo).