XMLPipeDB User and Developer's Manual

Loyola Marymount University
Bioinformatics Group

August 3, 2006

Contents

1 Overview

XMLPipeDB is a suite of tools for managing, querying, importing, and exporting information to/from XML data based on some specific XML schema (XSD). While its applicability is fairly general, the original motivation for XMLPipeDB is the management of biological data from different sources. Thus, XMLPipeDB's end-user applications are bioinformatics-oriented, although other applications may be developed using the project's underlying tools.

1.1 Who Should Read What Part of this Document

- If you are a GenMAPP user primarily interested in GenMAPP Gene Databases, Section 4.1 is for you.
- If you would like to *build* GenMAPP Gene Databases from supported XML data sources, you will want to read about GenMAPP Builder in Section 4.2.
- If you are a bioinformatics software developer who would like to create applications using XML data sources that have been imported into a relational database, you will want to read Section 3.
- If you are a software developer or database designer who is interested in learning how XMLPipeDB's end-user applications were developed, and would like to new Java libraries for working with XML data sources within a relational database, you will want to read Section 2.
- Finally, if you are a software developer who wishes to participate in or contribute to the XMLPipeDB project, you will want to read Appendix A.

2 Developer Tools

Many components in XMLPipeDB's end-user applications are actually built semi-automatically. The XMLPipeDB project includes these tools as well. End-users don't need to know about these tools; instead, they use the *output* generated by them. This section is for database designers and software developers who are interested in creating their own database applications using XML data sources.

2.1 XSD-to-DB

The XSD-to-DB application takes a well-formed XML Schema (XSD) or Document Type Definition (DTD) file and converts it into a collection of Java source code and Hibernate mapping files that allows XML files based on that XSD or DTD to be read into a relational database. It was used to help create the UniProtDB and GODB libraries which are used by GenMAPP Builder.

XSD-to-DB's conversion functions are based on the open-source Hyperjaxb2 project (`https://hyperjaxb2.dev.java.net`). It requires the following information to do its work:

- URL for the XSD or DTD file to process; in the case of a locally-loaded XSD or DTD, a `file:///` URL may be used
- Hyperjaxb2 binding file
- Directory that will contain its output

XSD-to-DB comes with a default binding file, which it copies into its output directory when it is invoked for the first time. Additional customization can then be performed on the copied binding file, and subsequent invocations of XSD-to-DB will use that binding instead of the default.

2.1.1 Usage

XSD-to-DB is currently packaged as a command-line Java application in an executable JAR, `xsd2db.jar`. As such, it can be invoked in a number of ways:

- Direct JAR invocation via `java -jar xsd2db`
- Running one of the provided shell scripts (i.e., `xsd2db` or `xsd2db.bat`)

In both cases, command line options are expected, described below.

`--xsdURL=`*`url`* The URL for the XSD to convert; required when XSD-to-DB is run for the first time, or when *updateXSD* (see below) is requested.

`-dtdSchema` Switch that tells XSD-to-DB to expect a DTD file in the provided `--xsdURL`.

`--outputDirectory=`*`dir`* The directory to use when generating (or re-generating) the database source code and files; defaults to `db-gen`. Specifying a non-existent or empty directory essentially counts as a "first-time run" of XSD-to-DB.

`--bindings=`*`filename`* The bindings file to use when generating the database source code and files for the first time; this file is then copied into the `xsd` subdirectory. Defaults to a standard bindings file supplied by XSD-to-DB.

`-updateXSD` Replaces the XSD being used with a new version; applicable only after XSD-to-DB has been run for the first time.

`-help` Displays a help message for how to use `xsd2db` (roughly a summarized version of this option list).

2.1.2 What XSD-to-DB Does

If you invoke it without any arguments, XSD-to-DB will issue a message requiring the URL of the XSD or DTD file to convert. Once the URL has been provided, XSD-to-DB then processes it produces Java source code, Hibernate mapping files, and an SQL DDL file that corresponds to the schema defined by the XSD file. These files are placed in an output directory, which defaults to `db-gen` if it isn't otherwise specified.

XSD-to-DB behaves a little differently depending on the presence of certain files in the output directory, or even the presence of the output directory itself. If the output directory is not present, then XSD-to-DB does the following:

1. Create the output directory.
2. Copy the XSD file from its given URL to the output directory.
3. Copy the default XSD-to-DB bindings file to the output directory.

If the output directory already exists, then XSD-to-DB looks for the XSD and bindings files at the expected locations within that directory. If those files are not present, then it performs the same steps as before.

- XSD-to-DB can be told whether or not to update the XSD file from its URL, in case the XSD file might have changed. If it is told to perform an update, it will ask for the URL of the updated XSD file.

Once the output directory has been set up, XSD-to-DB then processes the XSD and bindings files to generate:

- Java source code for classes represented in the XSD
- SQL DDL file defining the relational database tables that correspond to the Java classes
- Hibernate mapping files that determine how the Java classes are convert to and from the relational tables
- An Apache Ant `build.xml` file which can compile everything into a Java archive, ready for further development or deployment

From this point, a typical workflow would be:

1. Build a relational database using the generated SQL DDL file
2. Build the database library using the supplied Ant file, then use that library to test XML import, queries, and other database functions

3. Edit the bindings file to customize, correct, or improve the Java classes, Hibernate mappings, and relational tables generated from the XSD file

4. Re-run XSD-to-DB to actually create the new files

Certain conversions might not be adequate even with extensive editing of the bindings file; in this case, the last resort is to manually edit the Java, Hibernate, and SQL files generated by XSD-to-DB. If this is done, be careful about re-running XSD-to-DB on this particular output directory — XSD-to-DB *always* generates the Java, Hibernate, and SQL files "from scratch," using the XSD and bindings files in the output directory.

Typically, once the generated files work as desired, the output directory becomes a software project in and of itself, to be edited, debugged, tested, and deployed like any other database library. The fact that it was initially generated by XSD-to-DB merely indicates that some time was saved in creating this database library as compared to a manual Java-to-relational implementation of the XSD file.

2.1.3 Output Details

XSD-to-DB's output directory structure is shown in Figure 1. Directories are shown in boldface, and files are shown in italics. Names in parentheses indicate placeholders for specific names that depend on the XSD being converted.

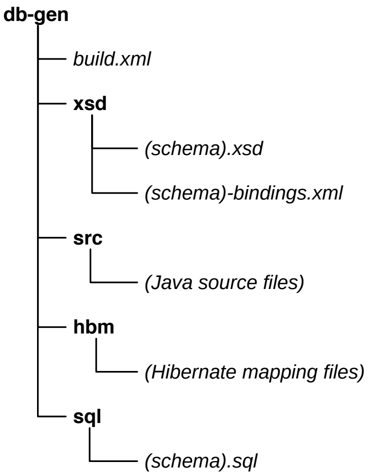


Figure 1: Structure of an XSD-to-DB output directory.

The actual filenames and contents of terms in angle brackets (<>) depend on the XSD file that was used to generate the output. For instance, the UniProtDB library has `uniprot.xsd`, `uniprot-bindings.xml`, and `uniprot.sql`.

2.2 XMLPipeDB Utilities

The XMLPipeDB Utilities library is a suite of Java classes that provide functions that are common to most XMLPipeDB database applications. Specifically, the library includes reusable classes for:

- Loading of XML files into Java objects, and subsequent saving of these XML-derived Java objects to a relational database
- Rudimentary query and retrieval of Java objects from the relational database
- Configuring a client application to communicate with a relational database

XMLPipeDB Utilities includes a sample GUI application that demonstrates these functions, using database code that was generated by XSD-to-DB.

2.2.1 Usage

XMLPipeDB Utilities is packaged in two JAR files: `xpdutils.jar` and `xpdutils-demo.jar`. `xpdutils-demo.jar` contains a demo BookDB application that shows how to use XMLPipeDB utilities. `xpdutils.jar` contains the utilities themselves; to write programs using XMLPipeDB Utilities, include `xpdutils.jar` in the Java classpath of the program you are writing.

2.2.2 What XMLPipeDB Utilities Provides

The library is separated into two layers: one layer provides the functionality, meant to be called programmatically, and another layer is a set of user interface components that allow end-user to invoke those functions. Figure 2 provides an overview of the library.

The XMLPipeDB Utilities library relies heavily on JAXB and Hibernate — in many respects, it is a convenience and GUI layer for these packages — so knowledge of these APIs is very helpful in making full use of XMLPipeDB Utilities.

Utility Functions The following classes implement the functions provided by XMLPipeDB Utilities:

- `ImportEngine` provides a *loadToDB()* method, which takes any compliant input stream, parses it into its corresponding objects, then commits the objects to the database.
- `ConfigurationEngine` is the centralized location for configuration information used by XMLPipeDB. It provides functions for retrieving, setting, and validating configuration properties. The ConfigurationEngine also has a method for obtaining a Hibernate Configuration object, which is needed by ImportEngine and QueryEngine. The configuration object, however, only contains the Hibernate properties and NOT

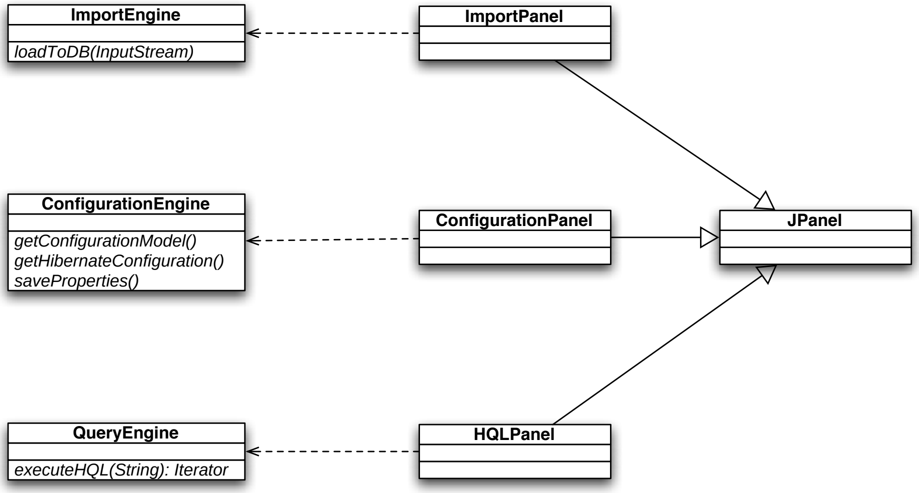


Figure 2: Overview of XMLPipeDB Utilities.

the mapping files. Hibernate mappings must be added by the caller before passing the Configuration object to the Query or Import Engines.

- `QueryEngine` is a generalized query wrapper whose *executeHQL()* function takes a Hibernate query (in HQL, Hibernate Query Language) and returns the results as a Java iterator.

Programs using the XMLPipeDB Utilities library can invoke these functions as necessary, using whatever mechanism is most appropriate for a particular application.

User Interface Components To ease the development and delivery of these functions to end-user applications, XMLPipeDB Utilities also includes a component library that can be added directly into Java Swing windows and panels. Each component provides a "hook function" that invokes the underlying operation, assuming that sufficient information has been gathered by the component:

- `ImportPanel` provides a front end for database imports, displaying components like file choosers, text editors, and preview panels to make database loading as easy as possible for end-users.

- `ConfigurationPanel` provides a front-end for the properties in `ConfigurationEngine`. The components are designed to be easily added to dialog boxes or preferences windows, and provide direct hooks to the `Configurator` functions.

- `HQLPanel` allows users to enter an HQL (Hibernate Query Language) or SQL (Structured Query Language) query then provides a table or object browser that displays the results of that query. This is meant primarily for debugging or advanced purposes, as it requires knowledge of the underlying database schema and/or domain object model. HQL queries are processed by `QueryEngine`.

The demo application that comes with XMLPipeDB Utilities shows how these components are used, and how they interact with the underlying functionality. GenMAPP Builder also uses XMLPipeDB Utilities for its import, configuration, and query functions.

3 Database Libraries

XMLPipeDB has used XSD-to-DB to generate Java database libraries for UniProt and Gene Ontology (GO) data sources, called UniProtDB and GODB, respectively. These database libraries use JAXB to convert XML files from these data sources into Java objects, then use Hibernate to persist these Java objects in a relational database, a schema for which is provided with the distribution. To use these libraries in other software, you will need the following in your Java classpath:

- the database library itself (`uniprotdb.jar`, `godb.jar`, etc.)
- Hibernate
- JAXB
- JDBC driver for your relational database

The libraries work like standard JAXB applications when importing XML files: the data in the loaded file gets converted into a set of corresponding objects. The libraries then work like standard Hibernate applications when saving these objects to the database: configure Hibernate, then use its classes to save/update the objects or to perform HQL queries.

To further assist in software development, you may want to look at XMLPipeDB Utilities (Section 2.2), which provide frequently-used routines and GUI components that are usually needed by applications using these libraries.

Relational schema diagrams for the database libraries are too large to include in the manual. They are available separately from the XMLPipeDB Web site (`http://www.cs.lmu.edu/~xmlpipedb`).

3.1 UniProtDB

UniProtDB is the library of Java classes that allows UniProt XML files to be transferred into a relational database. GenMAPP Builder uses the UniProtDB library; in turn, the UniProtDB library is semi-automatically generated from the XMLPipeDB project's XSD-to-DB tool.

3.1.1 Usage

To use UniProtDB in an application, follow these steps. Note that some familiarity with the JAXB and Hibernate APIs is required in order to use UniProtDB.

1. Include `uniprotdb.jar` in the Java classpath.
2. Initialize a relational database with the schema provided in `schema.sql`.
3. At the point where the application configures Hibernate, add `uniprotdb.jar` to the Hibernate configuration via the `addJar()` method. Also, make sure that Hibernate is configured for the relational database that was set up in the previous step.
4. Use the JAXB API to read UniProt XML files into Java objects, then use the Hibernate API to persist these objects to the relational database.
5. The XMLPipeDB Utilities library may be helpful in implementing some frequently-used functions such as XML import, configuration, and ad-hoc HQL querying.

3.1.2 Implementation Notes

XSD-to-DB's "as is" output from reading the UniProt XSD [WR04] required some post-processing prior to becoming usable in database applications. Specifically, the following changes needed to be made:

- The UniProt XSD defines an *end* element. Unfortunately, *end* is an SQL reserved word. Thus, the post-processor renames *end* to *endPosition*.
- Some XSD data types, particularly the range of options for XML dates, are not easily supported in SQL. In particular, XSD dates allow month-year values with unspecified dates, or even just years. This does not translate well to SQL and Java. It was concluded that values which were defined this way in the UniProt XSD can acceptably be represented as strings. This date mismatch occurred only once, in UniProt's *citationType* definition. *citationType*'s *date* attribute was defined as:

```
<xs:simpleType>
    <xs:union memberTypes="xs:date xs:gYearMonth xs:gYear"/>
</xs:simpleType>
```

- Another data type issue concerns strings. The libraries used by XSD-to-DB translated strings into SQL `varchar(255)`. However, many string values in UniProt XML files exceed 255 characters in length. Thus, instances of `varchar(255)` are converted into just `varchar` (unspecified length).

The UniProtDB post-processor is included in the `tools` subdirectory of the `uniprotdb` module in the XMLPipeDB CVS repository. Note that the post-processor only needs to be invoked if the UniProtDB files have been rewritten by XSD-to-DB. As committed, the files in the repository are already post-processed.

3.2 GODB

Information from the GO (Gene Ontology) database is required to run GenMAPP Builder. GODB provides a library of Java classes and Hibernate mapping (HBM) files that allows data from GO OBO XML files to be transfered into a relational database. The library is delivered and used as a Java Archive (JAR) file called `godb.jar`. The classes and HBM files generated by `xsd2db` are processed by a utility called `GodbPostProcessor` prior to being built into the JAR file. This is required because of some irregularities in the output provided by Hyperjaxb.

3.2.1 Usage

GODB includes a pre-built, ready-to-use copy of `godb.jar`. This may be used in GenMAPP Builder or in any other application you wish to build.

To use GODB in an application, follow these steps. Note that some familiarity with the JAXB and Hibernate APIs is required in order to use GODB.

1. Include `godb.jar` in the Java classpath.

2. Initialize a relational database with the schema provided in `schema.sql`.

3. At the point where the application configures Hibernate, add `godb.jar` to the Hibernate configuration via the `addJar()` method. Also, make sure that Hibernate is configured for the relational database that was set up in the previous step.

4. Use the JAXB API to read GO OBO XML files into Java objects, then use the Hibernate API to persist these objects to the relational database.

5. The XMLPipeDB Utilities library may be helpful in implementing some frequently-used functions such as XML import, configuration, and ad-hoc HQL querying.

3.2.2 Implementation Notes

XSD-to-DB's "as is" output from reading the GO DTD [Dat] required some post-processing prior to becoming usable in database applications. Specifically, the following changes needed to be made:

- The GO DTD defines an *to* element. Unfortunately, *to* is an SQL reserved word. Thus, the post-processor renames *to* to *to_*.

- Another data type issue concerns strings. The libraries used by XSD-to-DB translated strings into SQL `varchar(255)`. However, many string values in GO OBO XML files exceed 255 characters in length. Thus, instances of `varchar(255)` are converted into just `varchar` (unspecified length).

The GODB post-processor is included in the `tools` subdirectory of the `godb` module in the XMLPipeDB CVS repository. Note that the post-processor only needs to be invoked if the GODB files have been rewritten by XSD-to-DB. As committed, the files in the repository are already post-processed.

GODB needs to be regenerated if the GO OBO XML DTD changes. However, in this case, the utility provide to do the post processing, `GodbPostProcessor`, may also need to be updated, since new issues may be introduced with the GO schema change.

Regenerating GODB is a multi-step process:

1. Run `xsd2db`, providing the GO DTD URL as input

2. Run GodbPostProcessor on the `xsd2db` output.

3. Run ANT using the `build.xml` in the root of the GODB directory.

3.3 Database Administration and Setup

The applications and libraries in XMLPipeDB require any relational database for which a JDBC driver exists. Thus, they all require some degree of configuration, for specifying the database server, database name, username, password, etc. To facilitate this, XMLPipeDB includes a common database configuration GUI in all of its applications. Any other mechanism may also be used if desired, since database access is strictly “standard” JDBC and/or Hibernate.

Once a JDBC-ready database server has been successfully installed, the schema for the database needs to be instantiated within that server. XMLPipeDB modules provide one or more `.sql` files for doing this; executing the commands in those files within the database server instantiates the schema. In the case of UniProtDB and GODB, the schema file is called `schema.sql`. For GenMAPP Builder, the file is `gmbuilder.sql`

4 End-User Software

XSD-to-DB, XMLPipeDB Utilities, UniProtDB, and GODB are primarily for software developers. This section discusses the components of XMLPipeDB that are geared toward end-users, particularly bioinformaticians.

4.1 Pre-Built GenMAPP Gene Database Files

If your primary interest is to use GenMAPP for a particular organism, please consult the following list for GenMAPP files that have already been created using GenMAPP Builder (Section 4.2). In many cases, you can simply download them and point GenMAPP to them, and you'll be ready to load up expression data sets. Full details on GenMAPP are available on the GenMAPP Web site (`http://www.genmapp.org`).

- *Escherichia coli* K12

4.1.1 Overview of the GenMAPP Application and Accessory Programs

GenMAPP (Gene Map Annotator and Pathway Profiler) is a free computer application for viewing and analyzing DNA microarray and other genomic and proteomic data on biological pathways. MAPPFinder is an accessory program that works with GenMAPP and Gene Ontology to identify global biological trends in gene expression data. The GenMAPP Gene Database (file with the extension `.gdb`) is used to relate gene IDs on MAPPs (`.mapp`, representations of pathways and other functional groupings of genes) to data in Expression Datasets (`.gex`, DNA microarray or other high-throughput data). GenMAPP is a stand-alone application that requires the Gene Database, MAPPs, and Expression Dataset files to be stored on the user's computer. GenMAPP and its accessory programs and files may be downloaded from `http://www.GenMAPP.org`. GenMAPP requires a separate Gene Database for each species.

4.1.2 System Requirements and Compatibility

The Gene Databases produced by XMLPipeDB are compatible with GenMAPP 2.0 and 2.1 and MAPPFinder 2.0. These programs can be downloaded from `http://www.genmapp.org`. GenMAPP 2.0/2.1 and MAPPFinder 2.0 have the following requirements:

- *Operating system:* Windows 98 or higher, Windows NT 4.0 or higher (2000, XP, etc.)
- *Monitor resolution:* 800 × 600 screen or greater (SVGA)
- *Web browser:* Microsoft Internet Explorer 5.0 or later
- *Minimum hardware configuration:*
  - *Memory:* 128 MB (512 MB or more recommended)
  - *Processor:* Pentium III
  - *Disk space:* 300 MB disk (more recommended if multiple databases will be used)

4.1.3 Usage

1. Extract the zipped archive and place the `.gdb` file in the folder you use to store Gene Databases for GenMAPP. If you accept the default folder during the GenMAPP installation process, this folder will be `C:\GenMAPP 2 Data\Gene Databases`.
2. To use the Gene Database, launch GenMAPP and go to the menu item *Data > Choose Gene Database*. Alternatively, you can launch MAPPFinder and go to the menu item *File > Choose Gene Database*.

4.1.4 *E. Coli* K12 Gene Database Specifications

Gene ID Systems The *Escherichia coli* K12 Gene Database is "UniProt-centric" in that the main data source (primary ID system) for gene IDs and annotations is the UniProt complete proteome set for *Escherichia coli* K12, made available as an XML download by the Integr8 resource. In addition to UniProt IDs, this database provides the following proper gene ID systems that were cross-referenced by the UniProt data: Blattner, EchoBASE, and EcoGene. It also supplies UniProt-derived annotation links from the following systems: EMBL, InterPro, PDB, and Pfam. The Gene Ontology data has been acquired directly from the Gene Ontology Project. The GOA project was used to link Gene Ontology terms to UniProt IDs. Links to data sources are listed in the section below.

Species The Gene Database is based on the UniProt proteome set for *Escherichia coli* K12, taxon ID 83333. Two substrains of *E. coli* K12 have had their genome sequenced, MG1655 and W3110. This Gene Database contains data for the MG1655 strain (Blattner IDs). UniProt has a separate proteome set for the W3110 strain (taxon ID 316407). The W3110 strain uses IDs of the form "JWxxxx" and are not supported in this Gene Database.

Data Sources and Versions The current *Escherichia coli* K12 Gene Database was built on July 31, 2006; this build date is reflected in the filename `Ec-Std_20060731.gdb`. All date fields internal to the Gene Database (and not usually seen by regular GenMAPP users) have been filled with the build date. Versioning information for the individual data sources is given below.

- UniProt complete proteome set for *Escherichia coli* K12, made available as an XML download by the Integr8 resource:

  `http://www.ebi.ac.uk/integr8/FtpSearch.do?orgProteomeId=18`

  The XML filename is `18.E_coli_K12.xml`, downloaded as a compressed `.gz` file and extracted. Version information for the proteome sets can be found at:

  `http://www.ebi.ac.uk/integr8/HelpAction.do?action=searchById&refId=5`

The proteome set used for this version of the *Escherichia coli* K12 Gene Database was based on UniProt Knowledgebase release 8.3 on July 11, 2006.

- Gene Ontology gene associations are provided by the GOA project:

  `http://www.ebi.ac.uk/GOA`

  The gene associations are represented as a tab-delimited text file. The *Escherichia coli* K12 GOA file was accessed from the Integr8 proteome set download page:

  `http://www.ebi.ac.uk/integr8/FtpSearch.do?orgProteomeId=18`

  The GOA filename is `18.E_coli_K12.goa`, downloaded as a compressed `.gz` file and extracted. The GOA file for this version of the *Escherichia coli* K12 Gene Database was based on the July 2006 release of GOA (version information can be found as a date field within the GOA file itself).

- Gene Ontology data is downloaded from:

  `http://www.godatabase.org/dev/database/`

  We use the monthly release available at the first of every month. For this version of the *Escherichia coli* K12 Gene Database, we used the July 1, 2006 release. The file's name is `go_200607-termdb.obo-xml.gz`. We extract the file and reverse the period and hyphen in the filename so it reads `go_200607-termdb-obo.xml`.

Database Report

- UniProt is the primary ID system for the *Escherichia coli* K12 Gene Database. The UniProt table contains all 4329 UniProt IDs reported in the UniProt proteome set for this species.

- The Blattner IDs were derived from the cross-references in the UniProt XML for *Escherichia coli* K12. We compared our Blattner table with the table in the supplementary material from [RAA$^+$06], `Supplementary_Table_1_Annotation_E._coli_Genes.xls`. Our Blattner table contains 4466 identifiers. There are 219 Blattner IDs reported in the Riley et. al. table that are not in our Gene Database. Of these:

  - 157 are RNA genes (tRNA, rRNA, or misc_RNA)
  - 1 is the origin of replication
  - 51 are protein coding sequences (CDS)
  - 10 do not have a feature designation

  Conversely, there are 200 Blattner IDs in our table that are not in the Riley table:

  - 104 are in the form of "ECOK12Fxxx" and correspond to proteins encoded by plasmid F

  - 51 contain a period in the ID, such as "bxxxx.y"
  - 45 are IDs that have been supplanted by a new ID, but are reported in UniProt as synonyms to other Blattner IDs

- The *Escherichia coli* K12 Gene Database also contains 4156 EchoBASE IDs and 4224 EcoGene IDs that were cross-referenced by the UniProt XML.

- Unlike the official Gene Databases from GenMAPP.org, in the *Escherichia coli* K12 Gene Database, PDB has been designated as an "improper" gene ID system and cannot be used as an ID for gene objects on MAPPs. This action was taken because PDB IDs can refer to structures containing two or more different polypeptides and thus do not refer to a unique protein molecule.

4.2 GenMAPP Builder

GenMAPP Builder is an application for creating GenMAPP Gene Database files, or GDBs. These files are actually Microsoft Access database files (MDBs) with a `.gdb` filename extension.[1]

The application works by first importing supported data files into a relational database. The database can then be queried in order to produce a GenMAPP database file. At this writing, only one species may be imported into the relational database. Future versions of GenMAPP Builder will be able to store and export multiple species.

4.2.1 Supported Data Sets

The current version of GenMAPP Builder requires data from the following types of files in order to build a GenMAPP Gene Database file:

- UniProt XML

- Gene Ontology (GO) OBO XML

- Tab-delimited GO gene associations (various sources, primarily GOA)

Typically, the GO file need only be loaded once, since GO makes the entire set of terms available as a single download. UniProt XML files can be downloaded by species from the Integr8 Web interface.

Once the source files have been imported into the relational database, they are no longer needed; exports can be performed any number of times after that. In practice, however, periodic reloads will be necessary in order to remain in sync with the datasets' providers.

[1] GenMAPP handles three types of MDB files, distinguished by an extension other than `.mdb`: Gene Databases (`.gdb`), MAPPs (`.mapp`), and Expression Datasets (`.gex`).

4.2.2 Requirements

To use GenMAPP Builder, you need:

- A Java 5 runtime environment
- A relational database with an available JDBC 3 driver; however, at this writing, only PostgreSQL has been used and tested
- One or more data sets downloaded from the Internet, from data sources that are currently supported by GenMAPP Builder

Once these are properly installed and operational, you can perform these primary functions:

1. Import one or more supported data files into the GenMAPP Builder database
2. Perform pre-processing on the imported Gene Ontology terms
3. Build a GenMAPP Gene Database file based on a particular species in the database

The built files can then be read directly by GenMAPP.

4.2.3 Database Setup

Using GenMAPP Builder requires one setup step that is external to the GenMAPP Builder application: setting up the intermediate relational database that GenMAPP Builder uses to store imported XML information. This database is technically any relational database for which a JDBC 3 driver is available, although thus far only PostgreSQL has been used and tested.

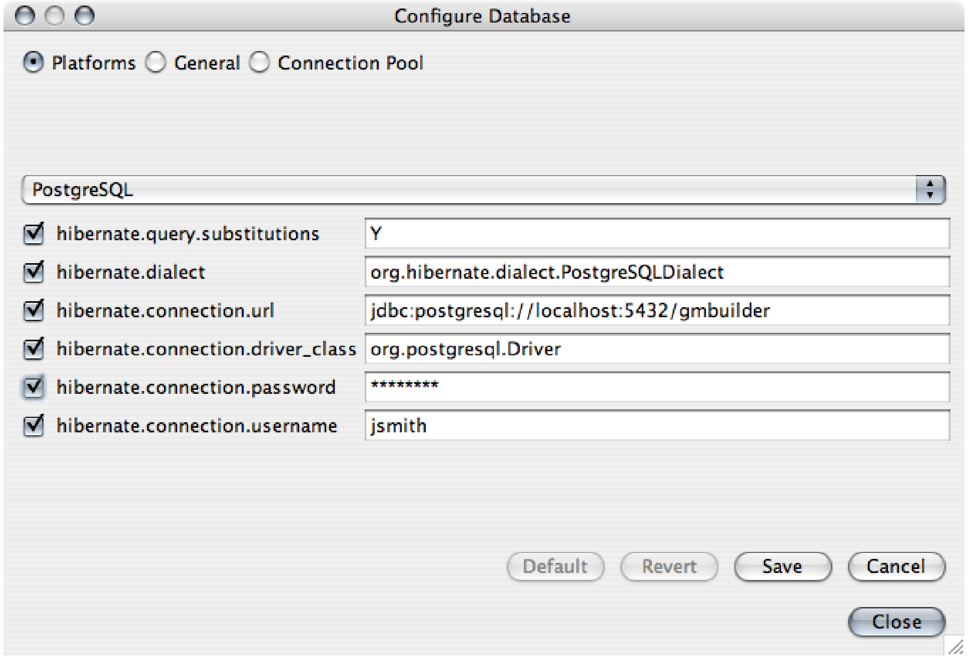
To set up the GenMAPP Builder intermediate database, perform the following steps:

1. Install the database server — this varies according to the specific database server software that you have chosen. For PostgreSQL, this involves installing the software, initializing a database cluster using the `initdb` program, then starting the server using the `pg_ctl` script or invoking `postmaster` directly.
2. Load the GenMAPP Builder schema file (`gmbuilder.sql`) into the relational database. Specifics for this step also vary depending on the database server that is used. For PostgreSQL, this step requires the creation of a named database using the `createdb` command, then importing the `gmbuilder.sql` file into that database through `psql`.

Once the database has been set up, all other configuration activities can be performed from the GenMAPP Builder application.

4.2.4 Configuration

When GenMAPP Builder is run for the first time, a configuration dialog appears (Figure 3). A valid configuration is required in order for GenMAPP Builder to operate properly; many problems are caused by incorrect database settings. The configuration can be changed at any time via the *Configure Database* command in the *File* menu.



● Platforms ○ General ○ Connection Pool

PostgreSQL

☑ hibernate.query.substitutions Y

☑ hibernate.dialect org.hibernate.dialect.PostgreSQLDialect

☑ hibernate.connection.url jdbc:postgresql://localhost:5432/gmbuilder

☑ hibernate.connection.driver_class org.postgresql.Driver

☑ hibernate.connection.password ********

☑ hibernate.connection.username jsmith

Figure 3: GenMAPP Builder configuration dialog.

The configuration dialog provides an interface for a wide variety of database-related settings; many of these can be left at their default values. The most important item to configure is the database server itself, a sample of which is shown in Figure 3:

1. Select the *Platforms* radio button.

2. Select the relational database that you are using from the drop-down menu.

3. Fill out the relevant settings from the list that appears.

The example shown in Figure 3 shows a GenMAPP Builder installation that uses a PostgreSQL relational database called *gmbuilder* on a server that is running on the same machine as GenMAPP Builder (*localhost*) using the default PostgreSQL network port, 5432. The standard PostgreSQL database driver is used (*org.postgresql.Driver*), and the database username is *jsmith* with an accompanying password.

Most of the time, these are the only settings to modify. Click the *Save* button when configuration is complete, then click *Close* to dismiss the configuration dialog.

GenMAPP Builder relies on the Hibernate library for its database interactions; a full reference of all settable parameters can be found on the Hibernate Web site (`http://www.hibernate.org`).

4.2.5 File Import

A "fresh" install of GenMAPP Builder contains no data — supported files must be *imported* into the relational database first. There is one *Import* command for each supported dataset, and these commands are all available under the application's *File* menu (Figure 4).

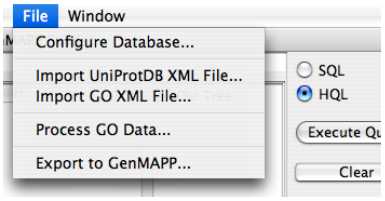


Figure 4: GenMAPP Builder *File* menu.

Each import command works in the same way: once chosen, a file chooser dialog appears. Locate the file to be imported through this dialog, then click on *Import*. The import process then begins. A dialog will notify you when the import completes; click the *Close* button to dismiss the import dialog.

Once an import operation finishes, the data in the imported file now becomes accessible as a set of relational database tables.

4.2.6 GO Pre-Processing

In the case of a GO OBO XML import, upon import completion GenMAPP Builder will ask if you would like to perform pre-processing of the imported GO data. This pre-processing step is required in order for Gene Ontology-related tables to be exported correctly into the Gene Database. Click *OK* to proceed with pre-processing, which may take around 30 minutes. A dialog will notify you when the GO pre-processing step completes.

The GO pre-processing step can also be performed at any time after a GO import takes place. Choose *File* > *Process GO Data...* to do this.

4.2.7 Browsing the Database

GenMAPP Builder uses the query engine from XMLPipeDB Utilities to allows you to browse the contents of the underlying relational database. With this engine, you can look at the species that are available and all imported information about the species' genes and proteins.

The query engine allows direct SQL or HQL queries, returning the results of these queries in a table or tree view (for SQL and HQL queries, respectively). The direct queries allow for maximum flexibility in examining the loaded data, but they require detailed knowledge of the intermediate relational database schema.

4.2.8 Building a GenMAPP Gene Database

Once the underlying relational database contains all of the information that you need (i.e., you have imported all of the XML data sets that you wish to export to a GenMAPP Gene Database), you can build a GenMAPP Gene Database file. To do this, choose the *Export to GenMAPP...* command from the *File* menu.

Definitions The following definitions are helpful in assimilating the information in this section:

- The term "system" by itself is a generic term referring to any gene ID system in the abstract (e.g., UniProt).
- A "systems table" generically refers to a table containing data from a particular gene ID system (e.g., the *UniProt* table in the GenMAPP Gene Database file is a systems table).
- *Systems* (capitalized, italicized) is the specific table used by GenMAPP to determine which gene ID systems (and thus, systems tables) are present in the Gene Database.
- A "relations table" generically refers to a table that relates two gene ID systems (e.g., *UniProt-EMBL* is a relations table that links UniProt IDs with EMBL IDs).
- *Relations* (capitalized, italicized), is the specific table used by GenMAPP to determine which gene ID systems are related in the database (and thus, which relations tables should exist).

Export Dialog The export operation opens with a dialog that requests the following information. Where possible, the destinations of these parameters in the final Gene Database are indicated.

- The owner of the database (limited to 200 characters): This goes in the *Owner* field of the *Info* table.
- The creation date of the database: This defaults to the current date, and goes in the *Version* and *Modify* fields of the *Info* table.

- The species to export: GenMAPP Builder examines the intermediate relational database and provides a drop-down menu for the species that are available there. Select the species for which you would like to export a Gene Database. GenMAPP Builder will automatically use this information wherever this is needed in the Gene Database, such as the *Info* table and the *Species* fields of the systems tables.

- One of these systems tables then needs to be designated as the *Model Organism Database* (MOD) system for the Gene Database. For a UniProt-centric Gene Database, this would be UniProt. The table name goes in the *MODSystem* field of the *Info* table.

- The location and name of the destination Gene Database file: As required by GenMAPP, the file has a `.gdb` extension. In addition, while not strictly enforced, the filename itself should follow this naming convention: two-letter code for genus/species, dash, the letters "Std" meaning standard or official, underscore, date in `yyyymmdd` format, file extension `.gdb`. For example, an *E. coli* Gene Database generated on June 6, 2006 should, by convention, have the filename `Ec-Std_20060606.gdb`.

- The relations tables to include: GenMAPP Builder automatically generates a default list according to an internal algorithm (specified in Section 4.2.10); the final list can then be modified (added to or deleted from) if desired.

- Systems table display order: An explicit ordering can be specified for the systems tables, determining the order in which the systems will display in the GenMAPP Gene Finder and Backpage. This information is stored in the *DisplayOrder* field of the *Info* table.

When these export parameters have been set, the export operation begins. Once the export operation finishes, the specified destination file can be opened and used by GenMAPP.

4.2.9 Implementation Notes

The configuration, import, and query functions in GenMAPP Builder make direct use of the XMLPipeDB Utilities library.

4.2.10 Database Details

GenMAPP Builder performs the following specific details in generating a Gene Database file that can be used by GenMAPP. Based on the chosen species, GenMAPP Builder first extracts the needed information from its intermediate relational database. Once this information has been retrieved, it is written to the specified Gene Database file as detailed in the rest of this section.

A sample GenMAPP Gene Database schema that results from the export operation appears in Figure 5. The schema in the figure is the result of a GenMAPP Builder export for *E. coli* with UniProt as the model organism system table.

1. The export begins by copying a blank GenMAPP Gene Database file, `GeneDBTmpl.gtp`, to the specified destination file (i.e., the one specified in the export dialog). The file is provided by GenMAPP.org, and consists of the following four tables: *Info*, *Systems*, *Relations*, and *Other*. The template file is actually a Microsoft Access database file with the extension `.gtp`. GenMAPP Builder comes with a default template file, but a different template file may be used since it may be changed periodically by GenMAPP.org.

2. Gene Ontology-related tables are then created, based on the system table that has been designated as the MOD for the exported database. These tables are *GeneOntologyCount*, *GeneOntologyTree*, and *MODSystem-GOCount* (e.g., *UniProt-GOCount*, in the case where *UniProt* has been designated as the MOD system table).

3. The *Owner* field of the *Info* table is filled with the *Owner* text given in the export dialog.

4. The *Version* and *Modify* fields of the *Info* table are filled with the date given in the export dialog.

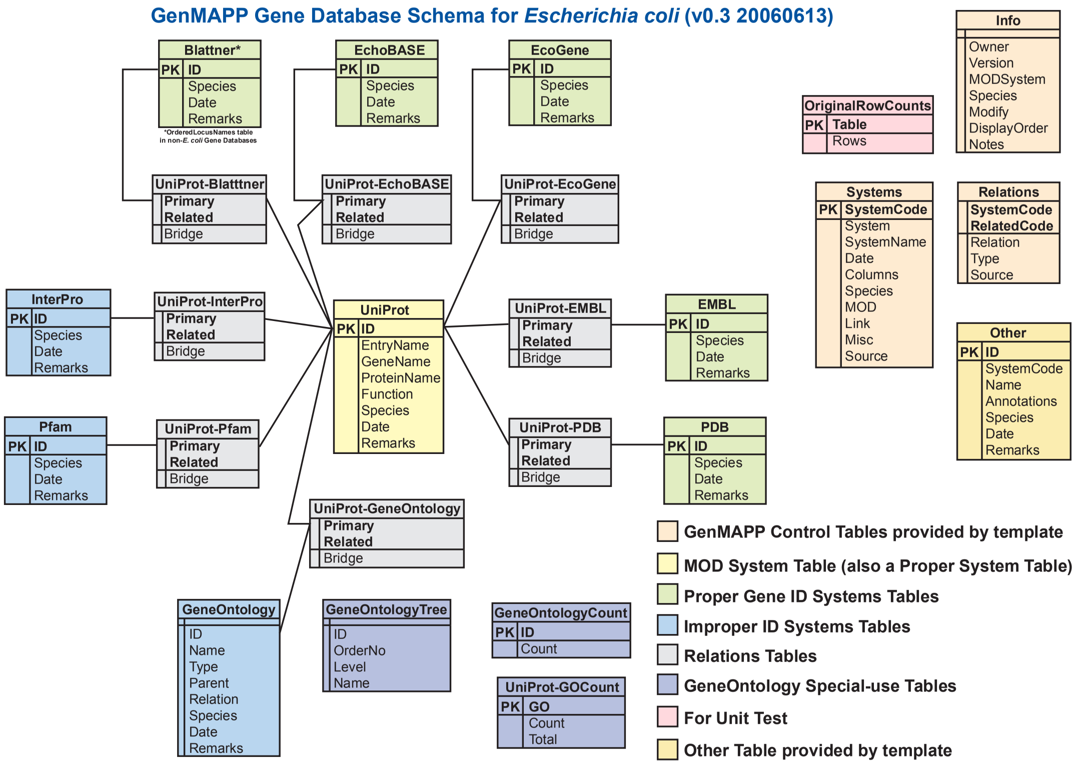
5. The *Species* field of the *Info* table is set to the genus-species designation given in the export dialog.

6. The systems tables specified in the export dialog are created. For UniProt-centric Gene Databases, at a minimum the *UniProt* table is created according to the schema shown in Figure 5. Other systems tables take on the schema of the *EMBL*, *PDB*, *InterPro*, etc. tables in the figure. Each of these tables has a corresponding row in the *Systems* table.

7. For each systems table that gets made, the *Date* field of the corresponding record in the *Systems* table is set to the current date.

8. The *OrderedLocusNames* table is created from the *ordered locus* name entries in the UniProt XML. Note, for *E. coli*, this table is renamed “Blattner” — an *E. coli*-specific post-processing step.

9. In the export dialog, the user was asked to designate one system as the MOD for the database. For UniProt-centric databases, it should be UniProt. The *MODSystem* field of the *Info* table is set to the name of this MOD system.



NOTE: Some Relations tables are not shown. All possible pairwise Relations tables exist between Proper ID systems and between Proper and Improper ID systems, but not between Improper ID systems (i.e., Proper-Proper, Proper-Improper, but NOT Improper-Improper).

Figure 5: Sample GenMAPP Gene Database for *E. coli*.

10. The *DisplayOrder* field of the *Info* table is set to the systems table order specified in the export dialog. The table sequence is specified by the abbreviated code (i.e., *SystemCode*) of each systems table delimited by the pipe character (e.g., "|S|Em|").

11. The relations tables called for in the export dialog are created. The default list of relations tables in that dialog is generated using the following algorithm: first, each system table is designated as being *Proper* or *Improper*, according to the *Misc* field in the *Systems* table of GenMAPP.org's existing Gene Database template file (the *Misc* field contains the string "|I|" if a systems table is improper). "Proper" systems tables contain IDs that refer to individual genes or proteins as objects in the database and therefore can be used as IDs for genes/proteins on MAPPs and in Expression Datasets. "Improper" systems store annotations that can refer to one or many different genes or proteins. Pairwise relations tables between Proper systems, i.e., all possible Proper–Proper, are created, with repetitions removed (e.g., if *UniProt-EchoBASE* has been added to the relations tables list, then *EchoBASE-UniProt* will not be added). Pairwise relations between Proper and Improper, i.e., all possible Proper–Improper, are also added to the relations tables list. No Improper–Improper relations tables are added.

    For example, the *E. coli* systems tables are classified as follows: UniProt, EchoBASE, EcoGene, and Blattner are proper, while EMBL, InterPro, Pfam, PDB, and GeneOntology are improper.

12. The template-provided *Relations* table is then filled with one record for each relations table in the exported Gene Database.

13. The *OriginalRowCounts* table is created and populated; this table can be used to verify that the tables are complete. The information in this table should correspond to the number of records for the chosen organism that were originally imported from XML into the intermediate relational database.

14. *Species-specific post-processing.* The variety of data sources available for a particular organism occasionally necessitates export operations that are specific to that organism. These activities comprise the final step of the export process.

    For example, in the case of *E. coli*, the following steps need to be taken:

    - In a generic UniProt-centric Gene Database, an *OrderedLocusNames* table would be created. However, in *E. coli* this table is called "Blattner." Thus, for *E. coli* only, the *OrderedLocusNames* system name must be changed to *Blattner* in the *Systems* table.
    - Further, a historical artifact with Blattner identifiers results in the inclusion of multiple IDs in a single field in some cases (e.g., "b1964/b1965/b1966"). These IDs have to be separated into distinct records in the final Blattner table.

- An *E. coli* Gene Database has UniProt as the MOD system table. There are more annotation fields than are in the Ensembl-centric Gene Databases provided by GenMAPP.org; as a result, the *Columns* field of the *Systems* table's *UniProt* record should be set to:

  `ID|EntryName\sBF|GeneName\sBF|ProteinName\BF|Function\BF|`

A Developer Documentation

A.1 Use Case Model

Figure 6 shows the use case model for the various components of the XMLPipeDB project. These use cases are the basis for the activities described in this manual.

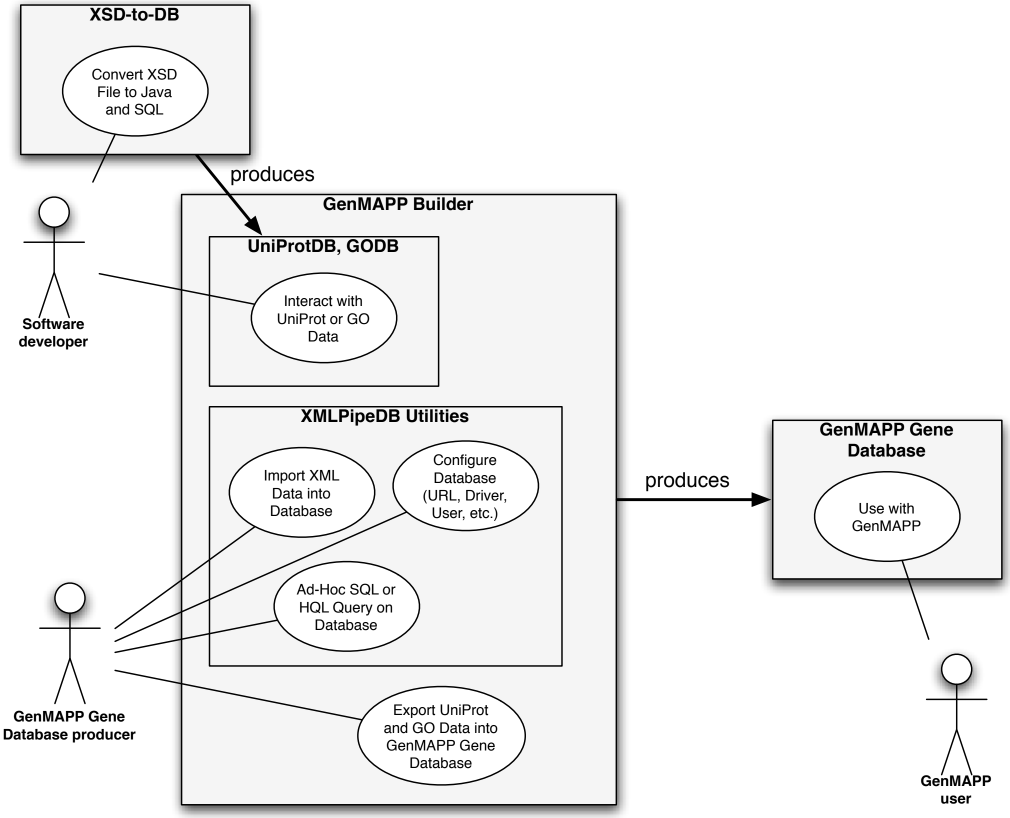

XSD-to-DB
Convert XSD File to Java and SQL
produces
GenMAPP Builder
UniProtDB, GODB
Interact with UniProt or GO Data
Software developer
XMLPipeDB Utilities
Import XML Data into Database
Configure Database (URL, Driver, User, etc.)
Ad-Hoc SQL or HQL Query on Database
GenMAPP Gene Database producer
Export UniProt and GO Data into GenMAPP Gene Database
produces
GenMAPP Gene Database
Use with GenMAPP
GenMAPP user

Figure 6: Use case model for the XMLPipeDB project.

A.2 Guide to the Repository

The XMLPipeDB project consists of a number of modules, listed hierarchically in Figure 7. They are related but intended to be standalone.

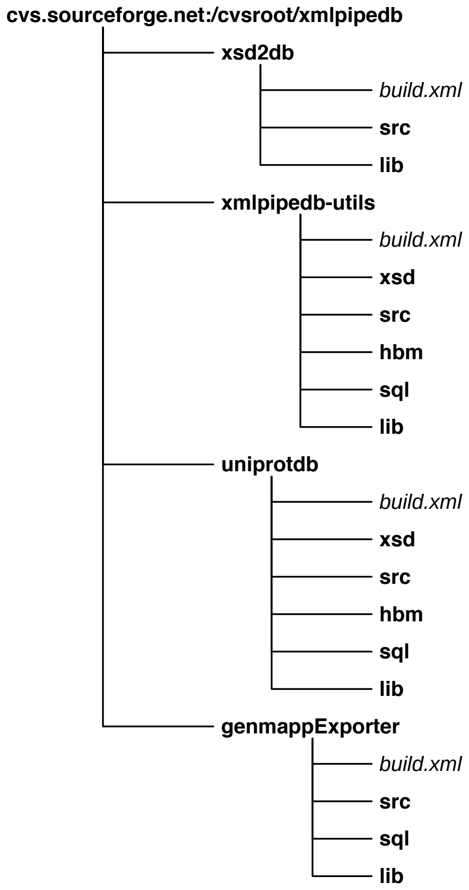


Figure 7: Structure of the XMLPipeDB CVS repository.

The `xsd2db` module contains the source code and libraries to XSD-to-DB. XMLPipeDB Utilities resides in the `xmlpipedb-utils` module, including the source for its demo application. When it is built, XMLPipeDB Utilities separates its reusable component (*xmlpipedb-utils.jar*) from the demo application that uses it (*xmlpipedb-utils-demo.jar*).

The sources for the UniProtDB library, which was generated by XSD-to-DB then manually customized, resides in the `uniprotdb` module. Finally, the source code to the GenMAPP Builder application resides in the `gmbuilder` module. Each module contains an Apache Ant `build.xml` file that can compile the source into distributable binaries; these build products are placed in the `dist` subdirectory, and should not be committed to the repository.

Similarly, output produced by XSD-to-DB (which defaults to `db-gen` in the current working directory at the time XSD-to-DB is invoked) should be committed as new modules in the repository, such as with `uniprotdb`. Of course, the default `db-gen` name should be changed to something more descriptive.

A.3 Building from Source

As mentioned, building fresh binaries of each XMLPipeDB module is a matter of downloading or checking out the source then invoking Apache Ant in each module's top-level directory. Build products are placed in the `dist` subdirectory, and should not be committed to the repository.

A.4 Third-Party Libraries

XMLPipeDB "stands on the shoulders" of a wide variety of third-party libraries. These libraries are always stored in the `lib/` subdirectories of each XMLPipeDB module. Most of these third-party libraries are themselves active open source projects that continue to be developed; thus, to avoid confusion and accidental incompatibilities, all third-party libraries committed to XMLPipeDB are appended with a version number. For example, if an XMLPipeDB module uses Jakarta Commons Logging 1.0.4, then `-1.0.4` is appended to the standard library name, so that it is stored in `lib/` as `commons-logging-1.0.4.jar`.

If a third-party library is to be updated with a new version — and of course this new version has been verified to work with the existing code — then its prior version should be deleted and a new file added with the appropriate version suffix.

B Support and Contact Information

- XMLPipeDB Gene Databases are built by the Loyola Marymount University (LMU) Bioinformatics Group using GenMAPP Builder. Both products are part of the open source XMLPipeDB project `http://www.cs.lmu.edu/~xmlpipedb`.
- For issues related to the *Escherichia coli* K12 Gene Database specifically, please contact:

  Kam D. Dahlquist, Ph.D.
  Department of Biology
  Loyola Marymount University
  1 LMU Drive, MS 8220
  Los Angeles, CA 90045-2659
  `kdahlquist@lmu.edu`

- For issues related to GenMAPP 2.0/2.1 or MAPPFinder 2.0, please contact GenMAPP support directly by e-mailing genmapp@gladstone.ucsf.edu.

References

[Dat] Gene Ontology Database. Gene ontology OBO XML DTD. http://archive.godatabase.org/latest/go_200607-obo-xml.dtd.gz.

[RAA+06] Monica Riley, Takashi Abe, Martha B. Arnaud, Mary K. B. Berlyn, Frederick R. Blattner, Roy R. Chaudhuri, Jeremy D. Glasner, Takashi Horiuchi, Ingrid M. Keseler, Takehide Kosuge, Hirotada mori, Nicole T. Perna, Guy Plunkett III, Kenneth E. Rudd, Margrethe H. Serres, Gavin H. Thomas, Nicholas R. Thomson, David Wishart, and Barry L. Wanner. Escherichia coli k-12: a cooperatively developed annotation snapshot. *Nucleic Acids Research*, 34(1):1–9, 2006.

[WR04] Allyson Williams and Kai Runte. XML format of the UniProt knowledgebase. In *12th International Conference on Intelligent Systems for Molecular Biology (ISMB 2004)*, Glasgow, Scotland, August 2004. http://www.ebi.uniprot.org/support/docs/uniprot.xsd.