Bio::Neos
2500 Crosspark Road
BioVentures Center #E108
Coralville, IA 52241
http://bioneos.com/

VCMap Documentation

Dr. Anne Kwitek (The University of Iowa)
Dr. James Reecy, (Iowa State University)
Dr. Mary Shimoyama (Medical College of Wisconsin)

This supplemental documentation is intended to assist in the execution, maintenance, and future development of the VCMap project, including the client application, webpage, and server load script. Bio::Neos will provide free support for any issues identified with the software for a period of up to 6 months after delivery of this project. After that period, support will be available at our standard hourly rate. Any requests for additional features or changes in scope of the project will not be covered under this support period.

Table of Contents

General Notes

The following documentation should aid in the maintenance and future development of the VCMap project. This documentation should supplement the comments and documentation already present throughout the source code for this project.

Subversion

The entire development history for this project has been recorded in a version control system called Subversion (SVN). It is strongly suggested that this system is used for future development as it will aid in incorporating changes from multiple editors, applying bug fixes to any release of the software (including older versions), and will provide insight into many of the previous changes (by reviewing the history logs). There are many sources for information about how to use this system on the internet, and versions that are written for all major operating systems.

The project repository is organized as follows:

/docs	Some initial discussion notes and project related documentation including posters and slides for talks.
/website	The actual VCMap launch website.
/src	The main directory for all of the project source code.
/src/layouts	Subdirectory for mockups and other testing code.
/src/client	Subdirectory for the VCMap client application.
/src/client/trunk	The main development branch for the client.
/src/client/tags	The previous tagged releases of the client.
/src/client/branches	The branches of the client codebase. Branches are created primarily to address bug fixes in previous stable releases of the client when development in the trunk has already advanced beyond the stable release
/src/server	Subdirectory for the VCMap server database load script.
/src/server/trunk	The main development branch for the server code.
/src/server/tags	(empty) Intended for branched releases of the server code when they are needed.
/src/server/branches	The branches of the server codebase. Intended for major modifications to the load script that should not be immediately placed in the primary development branch.

Ant

The two main `src` directories (`client` / `server`) contain `ant` build scripts (`build.xml`) with several targets. The default target should be suitable to build the necessary jars, but occasionally the `cleanbuild` target may be necessary to delete stale class files. Execute ant simply by typing '`ant`' from the command line prompt when at the sub-project root (`trunk` directory level, or a subdirectory of `branches` or `tags`).

API Documentation

The source code is thoroughly documented and well formatted. Additionally, comments are written in JavaDoc format so that a comprehensive API can be automatically generated for the current state of the source code. To do this, execute the following command from the trunk directory for either the client or server subdirectories:

```
ant api
```

This will create a set of HTML documents in the directory "`<trunk>/api`" designed for viewing with a web browser that supports frames. Simply open the file "`<trunk>/api/index.html`" and you will be given a list of the packages and classes defined for this project.

Dependencies

This project relies on the following third party code or applications:

- Ant (http://ant.apache.org/)
- Java SDK 5.0, or newer (http://www.oracle.com/technetwork/java/javase/index.html)
- Any webserver. Apache's HTTP Server is the recommended choice.
  Apache HTTP Server (http://httpd.apache.org/)
- Any SQL-based database. MySQL is the recommended choice.
  MySQL (http://www.mysql.com)

The following applications are highly suggested as they will improve the development environment and

- Subversion (http://subversion.apache.org/)
- Eclipse (http://www.eclipse.org/)
- Subclipse (http://subclipse.tigris.org/)

Support for these third party applications are outside of the scope of this document, but may be provided when deemed appropriate by Bio::Neos. **Please install these applications before seeking support from Bio::Neos.**

This documentation assumes that the server supporting this project is running a modern version of any linux distribution with the webserver and database systems installed and configured properly. Although "Linux" is not a required environment for this project, the documentation is written under the assumption that the server environment is such. The client system environment should be OS agnostic as the Java WebStart technology should make it possible to launch this system on any machine with a compatible Java Runtime Environment (JRE 5.0+).

Annotations

A consistent annotation system was used throughout the source code for this project. The following annotations indicate areas of importance throughout the code:

`FIXME`	This indicates an area of the code that is either not functioning correctly, or inefficient.
`TODO`	This indicates that an area of the code may not be totally completed or describes a useful feature that should be considered for inclusion. Typically these areas of the code will not affect usability, but often may affect robustness.
`DOC`	This simply indicates that documentation is needed.
`NOTE`	These sections provide a detailed description of an area of the code that may be complicated or otherwise difficult to understand without the information contained in this comment.

The first three annotations listed often indicate that some additional work is needed in this area of the code, whereas the `NOTE` annotations simply are important comments that should be reviewed before modifying that area of the code for any reason. The order of importance of the first three annotations is as follows:

```
FIXME > TODO > DOC
```

As suggested, the most critical issues are associated with the `FIXME` annotation, and the lowest level with the `DOC` annotation. When beginning future development for the project, it may be useful to review this annotations and determine whether or not they need to be addressed prior to beginning the implementation of new features. In addition, the `NOTE` annotations typically do not reference necessary code changes, but rather indicate more detailed and important comments than standard code commenting. When modifying or debugging code, pay close attention to all of these annotations as they will typically convey important developer notes.

Eclipse

The code repository has been designed to be checked out directly into an Eclipse project. Using the 'Subclipse' plugin, you can browse the repository and check out the code into a new or existing project. The Eclipse metadata is saved as hidden files in the repository so the project should automatically recognize the source code and enable formatting and code highlighting. Compilation of the project is performed using Ant in an external build instead of the internal Eclipse Java compiler. There are external build scripts included in the projects as '.launch' files.

Simply check out the `trunk` (or any subdirectory in `tags` or `branches`) of either the server or client subdirectories in the Eclipse IDE.

Server Database Load Script

The VCMap project relies on a relational database populated with comparative genomic data including annotation and positions. This data is gathered from several sources and compiled into a single database utilized by the client application to visually represent the data. The database has a supporting load script that performs all of the functionality related to the initial population of the database and the testing for, and gathering of, updated information. This load script is configured by text based, Javascript Object Notation (JSON) formatted configuration files. These configuration files allow for the reconfiguration of the script without any changes to the source code. The final delivery will contain the necessary configuration files to execute the script as configured for the data decided upon during the original project planning.

As of the completion of this document, the latest version of the client application is `v3.1.0`.

Building the Project

The source code for the client application is located in the Subversion repository as described in the General Notes section of this document. A checkout of the `trunk` directory or any `tags` or `branches` subdirectory will create a working directory containing the full project. There is an ant build script located at within these directories so the jar file can be compiled simply by executing the command `ant`, from the command line. Alternatively, if you check the project out using the Eclipse IDE, you must configure the External Tools to launch the ant build script (`build.xml`). Simply define a new Ant Task for this build file, and set the launch target to `vcmap-loader` (default). An example of this External Tools configuration is checked into the project (`VCMap Server compile.launch`).

Configuration Files

The load script behavior is defined by the configuration files. Namely, what species will be loaded and the specifications of the data source from which to retrieve that data. These files are written in JSON format (http://json.org/) and are fairly lengthy but relatively verbose. Included in the final package delivered at the end of this project are copies of the configuration files needed to load all of the data as agreed upon in the design and planning phase of this project (Mouse, Rat, Human, Chicken, Pig, Cow, and Horse). These files have been formatted with newlines and indentation to make it easier to examine and modify them.

There is one main configuration file that references the data files for the different data sources, and separately identifies the configuration files for the UCSC synteny and NCBI Homologene data, as these two files are in different formats.

The formats for the four types of JSON configuration files are listed in a later section of this document (`Main`, `Genomic Source`, `Annotation Source`, `Homology`, `Synteny`).

Program Flow

Described below is the process followed by the Server Database Load script to process and load data into the database. This outline is a simplified overview of the steps taken by the script to give a general overview of the process. Each item lists the methods that are critical for completion of

the described task. Review the source code and API documentation for more specifics.

1. Handle all command line options and setup initial operating environment.
   `bioneos.vcmap.db.VCMapLoader.main()`
   `bioneos.vcmap.db.VCMapLoader.setupLoaders()`
   `bioneos.vcmap.db.VCMapLoader<init>`
2. Main loop:
   `bioneos.vcmap.db.VCMapLoader.loadDatabase()`
3. Delegate the gathering / initial processing of each configured data source.
   `bioneos.vcmap.db.loaders.Loader.processData()`
   `bioneos.vcmap.db.loaders.*` (all sub-classes in this package)
4. Parse and load data from processed files.
   `bioneos.vcmap.db.VCMapLoader.loadData()`
   `bioneos.vcmap.db.VCMapLoader.loadAssemblyData()`
   `bioneos.vcmap.db.VCMapLoader.loadAnnotationData()`
5. Rebuild all "links" in the database from homology and direct reference data.
   `bioneos.vcmap.db.VCMapLoader.processLinks()`
6. Load all synteny data.
   `bioneos.vcmap.db.VCMapLoader.processSynteny()`

Step 3 of this process relies on the fact that the subclasses of the `bioneos.vcmap.db.loaders.Loader` interface all gather and process data into one of two format that can be handled by the main load script in a generic fashion. These formats are `tab-delimited` for assembly information, and `GFF3` (http://www.sequenceontology.org/gff3.shtml) for any annotation data. Once the data has been translated into these formats, the `VCMapLoader.loadAssemblyData()` and `VCMapLoader.loadAnnotationData()` methods can each respectively handle genomic and annotation data in a generic fashion.

Step 5 of this process builds a table in the database that comprehensively links homologous features across (and within) species. This table is rebuild every time that the load process is executed by truncating the table and regenerating the links. Homologous links are inferred in one of two ways: 1) An entry in Homologene ties features based on their NCBI GeneID, 2) Any features from the same source, with a matching identifier field are assumed to be homologous (or possibly ambiguously mapped). The second method allows for tying together features such as UniSTS markers that are mapped to both genomic and genetic maps. Potentially, enhanced processing can be added to this step more robustly link homologous data throughout the system, allowing for additional homologous links to be visualized by the VCMap client application.

Note: The load script does expect that all `GFF3` formatted files produced by the `Loaders` do not contain any features that reference `IDs` of other features. This is allowed in the official `GFF3` specifications, but in order to simplify our load process, all post-processed files are expected to have a unique `ID` for each row in the file.

Directory Structure

The 'scratch directory' as defined in the main configuration script is used by the main `VCMapLoader` class and all `Loader` subclasses in a consistent manner. In the `<scratch>/processed` subdirectory, there is a naming convention that must be strictly enforced. Outside of this subdirectory, any conventions are optional (although existing code always follows the same conventions). The naming convention is as follows:

```
/processed/<datasource_type>/<datasource>/
      <species>/<map_type>/data-<source>:<identifier>.gff3
```

Where:

- `<datasource_type>`
  Either genomic or annotation.
- `<datasource>`
  The name of the data source, typically where the files are hosted (Ex] NCBI UniSTS)
- `<species>`
  The species of the data.
- `<map_type>`
  Supports Genomic, Genetic, or RH. For annotation sources, this refers to the map for which the data will be loaded.
- `<source>`
  The source that actually generated the data (Ex] MARC). Note that this may be different than `<datasource>` because the files may be hosted at a different site than the group that originally generated the data.
- `<identifier>`
  The identifier for the data, indicated by the group that created the data not the host of the data. For example, for NCBI hosted genomic data this identifier refers to the string defined by the groups that generated the data, not the NCBI Build / Version strings.

Handling new Data Sources

In order to integrate a new data source into the system, you will need to subclass the `bioneos.vcmap.db.loaders.Loader` interface. The interface is straightforward and you can look at the existing classes within the `bioneos.vcmap.db.loaders` package for examples as how to implement this subclass. The basic contract of a subclass of this interface is that it will download any and all necessary data into the 'scratch directory' and then translate it into either a tab-delimited (genomic data source) or GFF3 formatted (annotation data source) file, and place this file in the correct location (see the section on Directory Structure above). Once the files are placed in the correct location, and have been configured with a proper JSON configuration file, the `VCMapLoader.loadData()` method should automatically identify and handle these data files.

Command Line Usage

The database load script is intended to be executed from a command line shell as a Java 'jar'file.

Executing the script without any parameters, or with any incorrect parameters, will print a usage description on standard output. The proper usage for the script is as follows:

```
java -jar vcmap-loader.jar [options] <config>
```

The `config` parameter takes the following form:

- `-c <file>` or `--config=<file>`
  Specify the main configuration file that specifies the general options for the loader script. This file should also refer to all of the various other configuration files using the 'source' attribute. **This option is required.**

The possible optional `options` parameters include:

- `-d` or `--devel`
  Operate in development mode. This causes the application to load information into the DB_devel database. Also, this will force the logger into debug mode. Implies `--cache`.
- `--cache`
  Use cached files instead of connecting to public sources to gather information when possible. If no cached files are found, files will be downloaded. No files will be deleted upon exit of the application.
- `--database=<db>` or `--db=<db>`
  Use `<db>` as the main VCMap database instead, overriding the setting in the config files.

Utility modes (These modes will perform a separate task from the database load, and exit immediately upon completion of that task. The typical data processing does not apply):

- `--insert-ontology=<file>`
  Use the OBO formatted file `<file>` to insert an ontology tree into the specified database.
- `--delete-map=<map-id>`
  This utility operation mode will cause the program to remove all data including synteny, versions, annotation_sets associated with the specified map with DB identifier `<map-id>`. This option will prompt you to confirm the delete before executing.
- `--delete-aset=<annotation-set-id>`
  This utility operation mode will cause the program to remove all data including versions and AVPs associated with the specified annotation set with DB identifier `<annotation-set-id>`. This option will prompt you to confirm before executing.

The script also depends on several third party libraries and compatible versions of these libraries have been included in the source code repository. Keep in mind that future versions of these libraries may or may not be compatible, so if they need to be upgraded for security reasons, or additional capabilities, make sure to test the entire project before starting.

Configuration Files Format

As described previously, there are 5 types of configuration files and they are all written in JSON format. The following listings will identify the proper format for these files and the optional and required properties of each format. Note that it is up to the Loader subclass to enforce these constraints and ultimately, the file format can be modified as long as the Loader supports all of the attributes used. Concrete implementations can add additional attributes to the configuration files, but cannot remove the listed required attributes; they are needed by the main `VCMapLoader` class.

Primary configuration file

```
{
  "scratch" : <required>        The location for temp files during the load process
  "processingDir" : <required>              The location for post-processed files
  "db_string" : <required>                  The JDBC style DB connection string
  "db_user" : <required>                                 The username for DB access
  "db_pass" : <required>                                 The password for DB access
  "logFile" : <required>                                  The filename for the log
  "homologene" : <optional>   The name of the homology (Homologene) config file
  "synteny" : <optional>               The name of the synteny (UCSC) config file

  "sources" : <required, array>                 An array of the filenames for the
  [                                                              datasource config files
    <string>,
    ...
  ]
}
```

Genomic Data Source

```
{
  "name": <required>                                              Data source name
  "type": <required>                                          Annotation or Assembly

  "species": <required, array of objects>
  [
    {
      "name" : <required>                                          Species name
      "taxID" : <required>                                 NCBI species taxonomy ID
      "maps" : <required, array of objects>
      [
        {
          "type" : <required>                              Genomic, Genetic, or RH
          "units" : <required>                                    Either bp, cM, or cR
          "scale" : <required>     Scale can specify how many digits of precision
                                this source uses.  This allows for decimal units to be
                                stored as integers in the database.  Use 10 to indicate
                                                tenths of precision, 100 to indicate hundredths.
          "version" : <required object>
          {
            "identifier" : <required>          The source assigned identifier
                                                for this data (can be set by the Loader
                                                implementation instead of hard coding)
            "release_date" : <required>  The date the source released this data
            "md5" : <optional>                Calculate an md5 checksum to track
                                                when changes are made to this data source.
                                                Alternatively, data changes can be determined
                                                by changes in the identifier or release_date.
          },
        },
        ...
      ]
    },
```

```
    ...
  ]
}
```

Annotation Data Source

```
{
  "name": <required>                                    Data source name
  "type": <required>                                    Annotation or Assembly

  "species": <required, array of objects>
  [
    {
      "name" : <required>                               Species name
      "taxID" : <required>                              NCBI species taxonomy ID
      "annotations" : <required, array of objects>
      [
        {
          "default" : <optional>          True or False.  The default annotation
                                            is loaded automatically when VCMap
                                          loads the map which references this data
          "type" : <required>               The type of feature (Ex] GENE or STS)
          "assembly_version" : <required>       The assembly_* fields combine
          "assembly_type" : <required>            to identify what assembly this
          "assembly" : <required>           data set should be associated with.
          "version" : <required object>
          {
            "identifier" : <required>             The source assigned identifier
                                                for this data (can be set by the Loader
                                              implementation instead of hard coding)
            "release_date" : <required>  The date the source released this data
            "md5" : <optional>                Calculate an md5 checksum to track
                                            when changes are made to this data source.
                                          Alternatively, data changes can be determined
                                            by changes in the identifier or release_date.
          },
        },
        ...
      ]
    },
    ...
  ]
}
```

The following two configuration file formats are largely tied to the data sources that are currently supported (Homologene and UCSC). When adding a new data source for a species that is supported by UCSC, remember to add a corresponding JSON object to the synteny configuration file. Outside of these additions, it is unlikely that these files will ever need to be changed, unless an entirely new way of gathering the homology data is implemented.

Homology

```
{
  "name" : "NCBI Homologene",                          This value is expected
  "type" : <ignored>                    Previously used.  Can now be safely ignored
  "hostname" : <required>       The host from which to grab the Homologene data
  "username" : <required>                              The username for that host
  "password" : <required>                          The password for that username
  "location" : <required>                      The location of the data on the host
  "filename" : <required>                          The name of the file on the host
  "columns" :                 This object implements a column index map as follows:
  {
    "HID" : <required>                    The column index of the Homologene ID
    "taxID" : <required>                  The column index of the Taxonomy ID
    "geneID" : <required>                 The column index of the Gene ID
    "geneSymbol" : <required>       The column index of the Gene Name / Symbol
  }
}
```

Synteny

```
{
  {
  "name" : "UCSC",                                     This value is expected
  "type" : "synteny",                                  This value is expected
  "threshold" : <required>   The smallest block (in bp) to load into the VCMap db
  "hostname" : <required>                          The host for the data source
  "username" : <required>                          The username on that host
  "levels" : <required, array of integers>             The "net" levels to load
  "synteny" :
  [
    {
      "name" : <required>                                  The species name
      "identifier" : <required>       The assembly identifier (in the VCMap db)
      "database" : <required>                The source species database name
      "table" : <required>       The target species database table (Ex] netHg17)
    },
    ...
  ]
}
```

Note: All of the `Loader` implementations can require any other configuration values that they need. The above listed parameters are only those utilized by the `VCMapLoader` class. A valid configuration file must include those values, or else have the `Loader` implementation set those values after processing the data files. For example, the `"version": { "release_date" }` value is required, but is automatically assigned by the NCBI based loaders after processing the data files from the source.

Additionally, all NCBI based loaders expect the following values within each annotations object:

```
"files" : <required, array of strings>
[
  <string>,                                  All files containing data from this source
```

```
  ...
],
"columns": <required>                    This object provides a map for the Loader
{                                          to grab data from a tab-delimited file.
  <string> : <integer>,                  The <string> identifies the type of data
  ...                                    and the <column> provides a column index
}                                          from which this data can be retrieved.
```

It is up to each concrete `Loader` implementation to enforce any rules on these required values as necessary.

Data Source Summary

The following data sources are queried as part of the data aggregation process for the VCMap Server load script.

Name	Type of Data	Species
NCBI Gene UniSTS Homologene	Genomic Annotation (Gene) Annotation (STS) Homology	Human Rat Mouse Cow Pig Chicken Horse
Ensembl	Annotation (Gene)	Human Rat Mouse Cow Pig Chicken Horse
ISU	Annotation (QTL)	Cow Pig Chicken
MCW	Annotation (Gene / QTL)	Human Rat Mouse
UCSC	Homology (Synteny)	Human Rat Mouse Cow Pig Chicken Horse

Database Schema

The database schema can be found in the repository at:

```
<repo>:/src/server/trunk/sql/VCMap.schema.sql
```

This text file clearly describes the database tables and references can be inferred from the naming conventions used throughout. For example, the `chromosomes:map_id` references the `maps:id` field, even though no foreign key constraints are defined. The schema file omitted these definitions because the data was defined for the MyISAM version 5.1 data table format, which does not store or use foreign key information: (http://dev.mysql.com/doc/refman/5.6/en/ansi-diff-foreign-keys.html).

Note: Basic source and external URL information is automatically inserted into the database by the final statements in this file. This data allows for automatic processing of the 'Source Link' and 'Homologene Link' hyperlinks within the VCMap client application. If additional data sources are added to the load script configuration, corresponding rows should be inserted into this file.

Future Improvements

The codebase is fairly feature complete, but there is one useful addition that you may want to consider adding to the system.

- We suggest that the execution of the script be placed on an automated cron job that runs once per month, in order to constantly update the database with the latest information. The load script should handle most data changes without issue, but some may require manual updating to certain sections of the configuration files. Therefore, after the execution of the script, it is important that the log file is reviewed for changes or updates that the script could not automatically handle, if email notifications are not implemented.
- Since the Server Load script is designed to run in an automated fashion, with a verbose log that can be long and difficult to review, it would be extremely useful to implement email alerts for important tasks. There are a few places in the code where some `TODO` annotations reference this capability. You can use the Java Mail package from Oracle to easily implement this feature.
- The addition of a `bioneos.vcmap.db.loader.Loader` sub-class that can handle data from the Ensembl Compara database as well as corresponding changes to the `processHomology()` and `processSynteny()` methods would allow you to combine homology data into a single source and get better coverage of the target species.

Developer Notes

If the script ever loads incomplete or incorrect data, or some archived data is no longer needed, you can always manually run the script in the utility modes, `--delete-aset` or `--delete-map`, to clean the specific data from the database.

While working on new features and updates to the load script and client application, it is suggested that you create a development database (named `<dbname>_devel`) and execute the scripts and clients in `--devel` mode to automatically connect to this development database. This will allow you to make and test changes (even database schema updates) before pushing the changes to the production version of the tool.

VCMap Client Application

The VCMap client application is a portable, cross-platform Java application launched via Java WebStart technology (http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/). The client loads comparative genomic data into memory via a direct connection to the server database, or through local data files opened by the user, and provides the user with an interactive visualization of the data.

As of the completion of this document, the latest version of the client application is `v3.1.1`.

Building the Project

The source code for the client application is located in the Subversion repository as described in the General Notes section of this document. A checkout of the `trunk` directory or any `tags` or `branches` subdirectory will create a working directory containing the full project. There is an ant build script located at within these directories so the jar file can be compiled simply by executing the command `ant`, from the command line. Alternatively, if you check the project out using the Eclipse IDE, you must configure the External Tools to launch the ant build script (`build.xml`). Simply define a new Ant Task for this build file, and set the launch target to `vcmap` (default).

Java WebStart Details

Because VCMap requests all permissions when executed on a remote system, Java WebStart enforces security restrictions including a required signature on all jar files and libraries deployed with the application. In order to sign these files, a keystore is required containing a certificate used to sign the code and verify ownership and integrity of the files within that jar. The most common issue with this method is that the security restrictions can often cause a situation where the application fails to launch with little explanation. This often is caused by one of two issues:

1. Incorrectly signed jar files
2. Improperly packaged third party libraries (missing `Trusted-Library: true`)

When signing the project jar files using the `jarsigner` command, make sure to sign all jars (even those from external parties) with the same certificate (the same keystore **and** alias). Avoid signing jars multiple times, as this causes problems with certain versions of Java. For this reason, all external jars are checked into the `lib` directory of the project without signatures. Take care to avoid checking in these files after they have been signed and placed in the `lib` directory of the `website` project.

For more information on the `Trusted-Library` issue, please see the `README.txt` file in the `lib` directory of the repository. This file provides clear instructions to avoid the issues associated with this tag and external libraries.

Deploying the Project

The client application is deployed in three easy steps:

1. Checkout the `svn-repo:/website` project to a web accessible location
2. Build the `vcmap.jar` file from the `svn-repo:/client` and place it in the root of the web accessible location you just created
3. Sign the `vcmap.jar` file, and all of the files in the `/lib` directory with the same certificate

Website

The website project (`svn-repo:/website`) contains all of the necessary files to deploy the application, as well as several informational webpages. As the project evolves you may want to update these pages to reflect updated news items, copyright dates, contact information, and more. The pages are all PHP scripts with similar content, but no advanced template or content management system has been put in place. The page does require a Javascript-enabled web browser in order to render correctly.

This project also contains the `.jnlp` files that describe the WebStart deployment parameters for the production and the development version of the application. These are named `vcmap.jnlp` and `vcmap-devel.jnlp`, respectively. You may need to edit these files as your needs change in the future. More information can be found on Oracle's website: http://docs.oracle.com/javase/1.5.0/docs/guide/javaws/developersguide/syntax.html.

Using the Autolaunch feature

The website project also contains an autolaunch script, `autolaunch.php`, that can be used to launch the VCMap application and automatically load some initial data into the interface. The autolaunch script accepts parameters via either an HTTP `GET` or `POST` operation and generates a custom `.jnlp` file for each request. The accepted parameters and their format are as follows:

backbone	<species>:<map-type>:<source>:<chromosome>[:version]
off-backbone[]	<species>:<map-type>:<source>:[:version]

Where:

- `species` (required)
  The name of the species as it is stored in the VCMap database
- `map-type` (required)
  Either `Genomic`, `Genetic`, or `RH`
- `source` (required)
  The name of the source for the map
- `chromosome` (required for `backbone`)
  The name of the chromosome in the format `chrXXX`
- `version` (optional)
  This can be omitted and the most current version of the map will be assumed.

The parameters should all be URL encoded (including the translation of the :colons into %3A) if the parameters are used in a GET operation.

This script generates the temporary .jnlp files in the <website root>/tmp directory. These files are small, but periodically should be emptied as the script does not handle cleanup of this directory.

Using --debug mode

We have implemented a special operation mode that should help with debugging and / or performance enhancements, named --debug mode. This mode instructs the application to record timing data for most of the major sections of the code, especially the graphical rendering routines. Once the application is launched in --debug mode, you will notice a new menu in the menubar called Debug. This menu can be used to display the current timing results, and to clear the totals while the application is running. In order to test performance of certain operations, the program can be launched and data can be loaded before clearing the timing totals and starting the operation. This mode of testing can be extremely useful for identifying inefficient areas of the codebase.

Additional timing tests can simply be added to the code using the bioneos.vcmap.VCMap.addTiming(String, long) method as well. Adding these method calls should only be done if the application is running in --debug mode, however, as they can incur a performance penalty.

Future Improvements

- Implement web services to replace the direct MySQL access. This will reduce security risks for the server (one fewer open port and application available from the Internet), as well as potentially solving some connection issues for remote clients that have very strict rules in place (not allowing outgoing connections on port 3306).
- The algorithm for grouping features allowing for useful lower resolution visualizations (zoomed out views) needs improvement. It is likely that the performance of this implementation can be improved, and the effectiveness of the algorithm is occasionally sub-optimal. The code for this algorithm is entirely contained in the method bioneos.vcmap.gui.MapNavigator.prepareDisplayMap(DisplayMap).
- The sections of the code that handle the ontology tree and filters is inefficient and poorly organized. There may be some TODO or FIXME annotations regarding this within the codebase. At some future point, this code should be improved.
- The autolaunch script is not currently utilized in external links. It would be beneficial to the users to make use of this feature more prevalently where links to the application currently exist or are being generated.