BioMart – User Documentation

Contents

Introduction

BioMart currently provides three user interfaces: the MartView web based interface and two java based user interface applications: MartExplorer, a graphical user interface and MartShell, a commandline textual interface. In addition there are two administration tools for creating and maintaining mart databases: MartEditor – a configuration editor and MartBuilder (in development) a tool for automatic database schema conversion to a mart format.

MartView

The original web interface to the BioMart schema based on the ensembl-mart perl library runs on an Apache/ModPerl server. It was originally integrated with the EnsEMBL website, but has been coaxed into a standalone system to provide access to other BioMart datasets. It facilitates the creation of a query using a three step 'wizard'. The user first chooses the focus dataset to query. In the next step, the user is given the chance to choose any filtering criteria to apply. This step also provides methods for getting count-based feedback for how many focus objects would be described in the results of a query with the given filtering criteria applied. In the third step, users can choose attributes of interest for the focus object, choose the output format, and execute the query. Attributes are partitioned in separate pages based on functional relationships, such that attributes on one page cannot be combined with those on other pages. In addition to the usage described above MartView also supports url-based batch querying. For examples see Appendix A.

MartExplorer

MartExplorer provides a graphical user interface that enables the user to quickly and easily construct queries and execute them against mart databases. Each query is built and modified in a separate QueryEditor, and the main window usually contains one or more of these. The QueryEditor contains three panels: a tree (top left), an input panel (top right) and a preview panel (bottom). The tree shows the current state of the query and can be used to change the contents of the input panel by clicking on the appropriate item: Attributes, Filters or Format. Attributes and filters can be added and removed using the appropriate input pages and the output format can be set via the format page. Attributes and Filters can also be deleted via the tree. Simply click on them and then press the delete key. Dragging and dropping them can also change the order of attributes, which sets the order of attributes in the output. Once the user is ready to execute a query, the results can be previewed in the preview panel, or all results can be saved to a file. It is easy to modify and rerun the query allowing for iterative development of complex queries. It is also possible to add and remove connections to other marts hosted on the network.

Running MartExplorer

MartExplorer can be started by running the script bin/martexplorer.bat or martexplorer.sh on windows and *nix OS, respectively.

Building and executing a query

When MartExplorer starts, the user is presented with a QueryEditor with some datasets to choose between. First, the user must select the dataset that the query is about. Once the dataset is selected, the user can select the attributes, filters and output format. The attribute panel is shown first but the user can switch to the filter or format panels by clicking on the Filters or Format items in the tree. The user can return to the attributes page by clicking the Attributes item in the tree. Once the user has built the query it can be run in one of two ways. It can be executed (Execute tool bar button or File -> Execute menu item) or the results can be saved (Save tool bar button or File -> Save / Save As menu items). Executing the query causes a preview of the results to be displayed in the in the preview panel, while saving the results causes all the results to be written to a file specified by the user. Results for the query can be saved to different files by using the File -> Save As menu item. Attributes and filters can also be removed, the attribute order changed, and the output format altered. There are two ways to delete items: (a) select the item in the tree and then press delete, (b) Remove it using the appropriate input page; for attributes this is done by unchecking the check box for the attribute in the data entry area. The order of the attributes in the results corresponds to their order in the tree. This can be changed by dragging and dropping them in the tree. Finally, the format can be changed via the Format page which can be shown by clicking on Format in the tree. Additional queries can be created by clicking the New query button, or selecting the menu item File -> New Query. Conversely, the currently selected query can be removed with File -> Close Query, and all queries can be closed with File -> Close All Queries.

Adding Mart Databases

It is possible to add new Mart databases in addition to the default ones already present when the application starts up. This can be done in two ways: by specifying an Mart Registry file and/or interactively during a MartExplorer session. The former makes added databases available during all subsequent sessions, the latter only for the duration of a single Mart Explorer session.

Registry file

At start up, MartExplorer reads a registry file which specifies a set of mart databases. The default registry file is located in the data directory: defaultMartRegistry.xml. It is possible to override its contents by placing a . userMartRegistry.xml file in the user's home directory. This must be a valid registry file. See the default file (data/defaultMartRegistry.xml) for an example, and the documentation for more details. The easiest approach is to simply copy the defaultMartRegistry.xml to user home directory changing its name to .userMartRegistry.xml, and edit the file according to one's requirements.

Interactively

Mart databases can also be added using the "Add Mart" settings window. This is opened with the menu item Settings -> Add Mart. This button opens the database connection window from which the user can specify mart database connection parameters. Once this is done, the marts will be available for use in new queries. Note that some values are "pre-loaded" into the drop down lists that are available by pressing the small downward arrow at the end of each text entry box. This can save you typing.

This is a brief description of the parameters together with some example values:

- Database type - Type of database server, e.g. mysql.
- Database Driver - This is the JDBC driver that the program uses to connect to the database e.g. com.mysql.jdbc.Driver.
- Host - This is the name of the computer on which the database server is running, e.g. martdb.ebi.ac.uk.
- Port - This is the port on the host computer that the database server is running on. The default port for MySQL servers is 3306.
- Database - This is the name of the mart database on the database server.
- User - This is the user name to use when connecting to the database, the user name "anonymous" works with martdb.ebi.ac.uk.
- Password - If the database requires a password then it can be entered here. MartExplorer can remember the password by checking the box next to the password field but it is stored unencrypted so this should not be used for sensitive passwords.
- Connection Name - This optional field can be used to give the mart database a user friendly name. The connection name is used in the dataset selection page of the QueryEditor. If the value is left blank then a default name is used.

Adding Registry and Configuration Files

Registry and Configuration Files can be added and removed interactively using the Add File settings window. This is opened with the menu item Settings -> Add File. The window contains a drop down list of loaded adaptors plus several buttons for adding and removing them:

- Add Registry File - This button opens a file selection dialog box from which the user can select a registry file. Once the registry file has been imported the marts it specifies will be available.
- Add Dataset Configuration File - This is only really useful for people developing their own dataset configuration files. The button opens a file selection dialog box from which the user can select a dataset configuration file. Once the file has been loaded the mart it specifies will be available for building queries. Note that it will not be possible to execute the query until a database has been associated with it. See Advanced Features for the method to change the database.
- Remove - Remove the file currently displayed in the drop-down list.
- Remove All - Remove all available files.

Advanced Features
With these features, the user can log output, and enable more options when constructing queries. Logging output can be enabled / disabled via Settings -> Advanced -> Logging. The log messages are printed to the console from which MartExplorer was started. Several advanced options can be enabled / disabled via Settings -> Advanced -> Enable Advanced Options. The advanced options are: a.) All datasets available in a mart become available when building queries rather than just the default ones. The system administrator who publishes the mart determines which datasets are default and which are not. Marts will often only contain default datasets.
b.) The mart database a query is executed against can be changed. This feature is useful for testing a query against several (compatible) datasets, and when developing a dataset configuration file locally and testing it against an existing database. The database can be changed by clicking on the Mart Database item in the tree view, and then selecting the database using the data entry area. c.) The dataset for a given query can be changed. The dataset configuration can be changed by first clicking on the Dataset item in the tree view, and then selecting the dataset in the input page that appears. All existing attributes and filters will be removed.

MartShell

MartShell is a commandline (text mode) Mart User Interface. The application is powered by a structured query language, called Mart Query Language (MQL). It provides capabilities for interactive inspection of Marts, and the Datasets they provide, and the execution of queries against these Marts. Queries can also be executed in batch mode using individual MQL command strings, or MQL scripts saved on the file system.

Commandline Options
MartShell provides commandline options to specify or change its behavior. The following is a list of commandline options available, with a description:

1. -h <command>: displays a list of commandline options with explanations, or, if a command is provided, help for the requested command (typing '-h help' provides a list of all applicable MartShell commands for which help is available).
2. -A Turns off Commandline Completion (faster startup, less helpful)

3. -R MARTREGISTRY_FILE_URL: URL or path to MartRegistry Bookmark document. The system first determines if a MartRegistry file called .userMartRegistry.xml exists in the users home directory, or, if absent, if data/defaultMartRegistry.xml file exists in the MartJ jar or distribution directory. If so, this will be the MartRegistry used to determine Marts and DatasetConfigs to load for the session, unless this is overridden with the -R swtich followed by the path to a MartRegistry.xml file.
4. -M SHELL_CONFIGURATION_FILE_URL: URL or path to shell configuration file. This file can contain a line of the form 'initScript = path'. This will instruct the system to run an MQL initialization script (see -I

below). If this switch is not provided, and a file of the same format named .martshell is found in the home directory, it will be used.

5. -I INITIALIZATION_SCRIPT: URL or path to Shell initialization MQL script. This file contains MQL statements (see section C.v.e.3 for information on MQL), but should really only contain statements predefining environment settings, Mart connections, DatasetConfig locations, or martRegistry files to add, stored procedures, etc. They should not execute MQL queries. The path to this script can also be specified in a shell configuration file, either $HOME/.martshell (recommended), or provided with the -M shell switch.
6. -d DATASET : Name of Dataset to use in queries for the martshell session (batch or interactive), unless overridden with other MQL statements. This can be of the same form as that specified in a 'use' or 'using' statement (see below, on the ‘use’ command for more information).
7. -v : verbose logging output, for debugging purposes.
8. -e: MARTQUERY : one or more MQL statements, separated by semi-colons and enclosed in single quotes (see Appendix B for more examples).
9. -E QUERY_FILE_FILE_URL: URL or path to file with valid MQL statements, or comment lines beginning with '#'. See Appendix B for examples. The following can be used in combination with the -e or -E flag:
10. -O OUTPUT_FILE : output file, default is standard out
11. -F OUTPUT_FORMAT: output format, either tabulated or fasta
12. -S OUTPUT_SEPARATOR: if OUTPUT_FORMAT is tabulated, can define a separator, defaults to tab separatedHelp: MartShell provides extensive documentation. Users can access this information from both batch and interactive modes. Help offers information on many MQL topics, and provides examples of MQL usage. The list of topics can be discovered interactively just by typing help, or in batch by typing '-h help' or using the -e switch with 'help' as the command. Help on a particular topic is available in interactive mode by typing 'help topicName;', and in batch mode either with '-h topicName' or using the -e switch with 'help topicName' as the command.

MQL overview

MQL is a structured query language (much likethe SQL used in querying a relational database). It provides methods allowing all of the operations within a BioMart User Interface, in a compact format, which resembles a human sentence. One or more MQL statements can be written along one line, or spread across multiple lines, with white-spacing to make the statements easier to read. All MQL statements must end with a semi-colon ';', unless a single MQL statement is being passed to MartShell from the commandline using the -e switch. Lines beginning with the '#' symbol are ignored as comments (more applicable to MQL scripts).

1. set Mart: Users can connect to any Mart hosted on their network. There are commands for adding and removing connections to Marts, adding DatasetConfig files for these Marts, listing available marts, describing a mart to which the user has connected, and setting the environmental mart for a session.
   (a) add Mart: This command can be used to instruct MartShell to connect to a Mart hosted on the network. The format of this command is 'add Mart <connection_parameters <as name> >;'. If in interactive mode, and no connection parameters are supplied, the user is presented with a dialogue to obtain the necessary information needed to establish a connection. Connection parameters consist of multiple key = 'value' (quotes required) statements, separated by commas. Connection parameters must be provided in an add Mart statement run in batch mode. Additionally, users can name a Mart connection, either by answering the appropriate question in interactive mode, or by ending the 'add Mart connection_parameters' statement with 'as name', where name is any string that the user will remember. This name will be presented to the user during 'list' requests, and must be used in 'set', 'use', or 'using' statements to refer to the Mart. It is highly recommended that users name their mart connections, as the default is hard to remember. Connection parameter key can be one of the following:
      - host: sets the RDBMS host (required)
      - port: sets the RDBMS port. Defaults to 3306 for mysql.
      - user: sets the RDBMS user (required)
      - password: sets the RDBMS password.
      - instanceName: sets the name of the Mart Database Instance to query (required).
      - databaseType: sets the type of RDBMS server. Note, if both databaseType and jdbcDriver are left blank, type ‘mysql’, and driver 'com.mysql.jdbc.Driver' are used, respectively.
      - jdbcDriver: sets the name of the Java JDBC Driver class to use to manage JDBC connections. Note, if both databaseType and jdbcDriver are left blank, type 'mysql', and driver 'com.mysql.jdbc.Driver' are used, respectively.- add datasets: This command allows users to load a MartRegistry.

The format of this command is 'add datasets file|url', where file or url must point to a valid MartRegistry.xml document.

b) list Marts: this command will list the names of all marts that the system has connected with, either specified explicitly by the user, or specified in MartRegistry or Initialization scripts. The names that are listed are either of the format user@host:port, or the name provided by the user when the Mart Connection was specified.
c) describe Mart: Lists the connection parameters (user, host, etc) for a Mart Host. The format of this command is 'describe Mart name;' where name is either the default (see list Marts, above), or the name provided in the specification (user or MartRegistry file).
d) set Mart: sets the environmental Mart to use for the rest of the session, unless overridden with another 'set Mart' statement, a 'use' statement (which also sets the environmental Mart, or a 'using' statement which sets the Mart for a single MQL query. The format is 'set Mart name;' where name is the name of the Mart. Note, while the name 'userFiles' is used like a Mart Name for the purpose of grouping datasetConfigs, this term is not a valid Mart, and cannot be used in a 'set Mart' command.

2. set Dataset: MartShell offers various commands to manipulate Datasets.
   a) add datasetConfig: this command allows users to add a DatasetConfig file from the file system. The format for this command is 'add datasetConfig file|url;'. Where file or url mustpoint to a valid DatasetConfig.xml file. These DatasetConfigs are grouped in the list of datasets under 'userFiles', and are not associated with any Mart to which the user has connected. To use these configurations, the user must use a special syntax in the 'use' or 'using' statement to associate the configuration with a Mart (see below for more information).
   b) list datasets: This command allows the user to list datasets that have been loaded, either from the Marts that the user has connnected, or from configurations loaded from the file system.The Format of the command is 'list datasets <all|userFiles|MartName>;' If no environmental Mart has been set with a 'set Mart' command, the user must type either 'all', the name of a Mart connection, or 'userFiles', otherwise the system will emit an error and message. 'All' lists all datasets as MartName.Dataset (eg, the format for specifying a particular datasetConfiguration in an absolute 'use' or 'using statement, see below for more information), where MartName could be 'userFiles'. 'userFiles' lists any datasetConfigs loaded from the file system as userFiles.Dataset. 'MartName' lists all Datasets loaded from a particular Mart Host as MartName.Dataset. If the environmental Mart has been set, 'list datasets;' will list all Datasets for the environmental Mart as Dataset (eg the format for specifying a particular Dataset relative to the environmental Mart).
   c) list datasetconfigs: Most Marts will provide only one, 'default' dataset configuration for each dataset that it hosts. In rare situations, a Mart could provide alternative configurations for the

same dataset. In these cases, users can only use these alternative configurations by specifying them completely in a 'use' or 'using' statement (see below for more information). The list datasets command only lists default datasets for a Mart. The list datasetconfigs command lists any alternative configurations loaded for a particular dataset as MartName.datasetName.configName. ConfigName could be any name, but the name 'default' is reserved as the name for the default configuration (see 'use' below for more information).

d) use: the use command can be used to set the environmental Dataset, or Mart and Dataset for a MartShell session. Its actions can only be overridden with a 'set Mart' command, another 'use', or a 'using' command. The command follows the format: use dataset_request where dataset_request specifies a dataset configuration, either relative to the environmental Mart, or absoulute. Absolute requests are of the form 'x.y.z' (x is the name of the mart or 'userFiles', y is the name of the dataset, and z is the name of an alternate configuration to use for dataset y); or 'x.y' (x is the mart name or 'userFiles', y is the dataset name, and the default dataset configuration for dataset y is used). Relative requests are of the form x (default configuration for dataset x) or 'x.y' (dataset x, configuraton y). If the environmental mart has not been set, then setting a mart with a relative name will throw an error. When using an alternate configuration that has been loaded from the file system, a special notation must be used to 'hook' this configuration to a particular mart. This is of either of the form 'userFiles.y.z>x' (configuration z for dataset y against mart x) or 'userFiles.y>x' (default configuration for dataset y against mart x).

3. set output Format: MartShell provides the set output command to set the output format for a Session. These will override any settings specified on the commandline. The format of this command is 'set output <output_parameters>;' where output_parameters is a list of one or key = 'value' specifications (values must be enclosed in single quotes, see below for available keys). If multiple key = 'value' specifications are provided, each one must be separated by a comma (eg, key = 'value', key = 'value'). Like the 'add Mart' command (see above), this command behaves differently in batch and interactive mode. In interactive mode, output_parameters are optional. If left off, the user is presented with an interactive dialogue for each possible setting, with the option of keeping the current setting for each individual parameter. In batch mode, the user must provide output_parameters. Keys for output_parameters can be one of the following:
    a. format: This sets the format for the output. Currently, only fasta and tabulated are supported, but more are planned in the future. Fasta is only applicable for sequence queries. Tabulated can be used for both sequence and non-sequence queries.
    b. file: This sets the file where query output is sent. This can be a file on the file system specified by 'path' or 'url', or it can be standard output, which is specified by a dash ('-').

c. separator: This sets the record separator for tabulated output. It is ignored for fasta output.

4. choose Filters/Attributes and execute: This process is actually encoded into a single command structure of the form (portions enclosed in angle brackets are optional):

<query_function>
<using dataset_request> get <attribute_list>
< sequence sequence_request >
< where filter_list >
< as namedProcedureName >

Query functions are strings that instruct the shell to further process the results of a query (see below for details). The using clause is optional, but, if it is not present, a Mart and Dataset must have been set with the 'use' or 'set' command, or the –d commandline switch to MartShell (see Appendix B for examples). Note, a using clause will not affect the environmental mart and dataset, but only override them for a single query. They are most useful for storing queries for later use. Attribute_list is a comma-separated list of mart attributes. These must match the internal_name of attributes in the DatasetConfig for the mart being queried (see below for commands to list attributes and filters). Attribute_list can be omitted for sequence_requests, otherwise at least one attribute must be specified. Specifying attributes with a sequence_request requests that those attributes (if available) are included, either as fields in tabulated output or in the description portion of the header in fasta output. filter_list is a comma separated list of Filter Requests. Filters must be in one of the following formats:

a) filter_name excluded|only -- specifies that objects should be returned only if they match (only), or only if they do not match (excluded) this filter. This format is only applicable to boolean and type filters.
b) filter_name (=, !=, >, >=, <, <=, or like) value
c) filter_name in list: List requests must be in one of the following formats:
d) filter_name in path|url - path/url must point to some resource with ids matching the filter_name, one per line
e) filter_name in (comma_separated_list of ids). Note, the list must be enclosed in perentheses, but no quoting is required
f) filter_name in namedProcedureName. filter_name must match the internal_name for a filter specified in the DatasetConfig for the mart being querried (see below for commands for list attributes and filters). When the filter is part of a filter_set (as specified in the DatasetConfig System), the filter_name must be further qualified with a filter_set_name using the filter_set_name.filterName notation. Multiple filter requests must be separated with the word 'and'. Using an 'as' clause stores the command as a named Procedure.In addition, users can use the 'list' command to discover attributes/filters that the environmental dataset provides. If the

environmental dataset is not set, an error is thrown. The format is 'list attributes' or 'list filters'.

5. Query Functions: Martshell provides the user with functions that take the results of a query and further process it. Currently, only count_focus_from is supported, but more are planned in the future. Instead of running a query, this counts the number of focus objects that would be returned by a given query, with its given filters (attributes are ignored). Its format is 'count_focus_from dataset_request| storedProcedureName|MQL;'. Providing a valid dataset causes the system to count the total number of focus objects provided by that dataset. Providing a storedProcedureName causes the system to count the number of focus objects that would be returned given the filters in the MQL associated with the name. Providing a complete MQL statement (see section C.v.e.3.d, but note, MQL cannot contain two query functions) counts the number of focus objects returned bythe query given its filters. Note, a query with no filters will return the same count as count_focus_from dataset_request.
6. Execute MQL scripts: The system provides a command for executing other MQL scripts. This is of the form 'execute script file|url;' where file or url points to a valid MQL script. This command can be used in both batch mode (even included in an MQL script), and interactive mode.
7. StoredProcedures and Query Chaining: One very powerful feature of MartShell is its support for query chaining, using stored, named MQL queries. Although support for query chaining is planned for all BioMart User Interfaces, MartShell is the only interface that currently supports the feature. Any MQL command can be stored as a named procedure by appending 'as name' to the end of the query, where name can be any word that the user can remember. It is strongly recommended that stored Procedure MQL be specified with a 'using' clause specifying their Mart and Dataset. Users might want to store useful Stored Procedure MQL commands in their $HOME/.martshell initialization MQL script. The name for all stored procedures that have been created are provided using the 'list procedures' command. Stored procedures can be executed individually using the 'execute procedure name' command. The name of a stored procedure specifying one attribute can be used to provide a list of data to be used in a list filter for another query. In this process, the system first executes the 'inner' query to create a list filter for the 'outer' query. The 'outer' query is then executed with this list filter in place. Currently, users must know which filter from one Dataset is applicable to a particular attribute in another dataset (or even in the same dataset). In the future, the system will provide 'implicit' query chain filter specifications whereby the user will not need to know how to create these connections, in addition to the traditional ‘exploratory' chaining whereby the user makes these connections explicitly.

MartShell Usage

MartShell has two modes of interaction:

1. Interactive Mode: As alluded to in the previous sections, MartShell provides different behavior when run in 'interactive' mode. Some commands provide interactive sub dialogues to help the user set environmental settings (add Mart, set output).
    a. help: Extensive online help is available to users with the 'help' command. The format for this is 'help <topicName>'. Typing help by itself provides a list of available topics. Typing 'help topic Name' prints help for a particular topic.
    b. tab-completion and commandline history: For many systems, MartShell provides context sensitive tab-completion, and an editable commandline history. Currently, this is only available for linux and alpha UNIX systems, but more are planned in the future.
        – tab completion: Hitting the 'tab' button twice quickly in succession at some point in a command (even at the beginning) will attempt to complete any partically typed word in the command. If it finds an exact match, it completes the word automatically. If it finds multiple potential matches, it presents the user with a choice of potential matches. If the user has not typed anything, a list of potential MQL command keywords is presented. If the user is in an MQL command, but is not in the middle of a word (eg, using x.y get <tab><tab>) a list of potential matches at that point in the command is presented. Particularly, when in the middle of an MQL query (even one spanning multiple lines) the system will present potential datasets after 'using', potential attributes when in the middle of the 'get' portion of a command, and potential filters when in the middle of the 'where' portion of the command.
        – history: MartShell provides the user with access to an editable commandline history. This history is saved between sessions. Each time a command is entered, it is saved onto the history. By typing the 'up-arrow' and 'down-arrow' keys, the user can scroll through the entire history. Once an interesting query in the history has been found, the user can use the 'left-arrow' and 'right-arrow' keys to move to particular positions in the command, use the backspace and delete keys to remove characters in the command at any spot, or even type new characters at a particular spot. MartShell also offers a 'history' command, which will show all, or a portion of the history. The format of this command is 'history <x<,y>>;'. Typing 'history;' prints out the entire history. Typing 'history 9;' prints the 9th element in the history. Typing 'history 9,;' prints out the entire history, starting from the 9th element in the history. Typing 'history,8;' prints the first 8 elements

of the history. Finally, typing 'history 3,20;' prints the 3rd through 20th element of the history. Users can also execute one or more history commands using the 'execute' command. This is of the form 'execute history <x<,y>>;'. See history above for an explanation of the x,y notation. Note, execute will ignore previous 'execute history' commands, to avoid infinite loops.

- saveToScript: this command allows users to save some or all of their history to another file (remember that the system already saves its entire history between sessions). The format is 'saveToScript x<,y> path|url;' This allows the user to save commands 10 to 15 to an MQL script using 'saveToScript 10,15;'.
- loadScript: This allows users to load MQL scripts into the history without executing them. The format is 'loadScript path|url;' where path or url points to a valid MQL script.

2. Batch Mode: Batch mode provides users with the capability to plug MartShell into other pipelines. There are two methods for runningMartShell in batch mode. There is also a batch 'help' mode, which provides help on all of the topics that are applicable to running MartShell in batch mode.
   a. help: Users can get help using the '-h' commandline switch. If used alone, the user is presented with a list of possible commandline switches. If a topic name is provided after -h, then help on that topic is printed. For a list of available topics, use '-h help'.
   b. -e commandline switch: This feature allows the user to provide one or more MQL statements, separated by semicolons, in single quotes after the -e switch. This feature is most useful for short queries, as most consoles limit the number of characters a user can type on a single command. To help with this constraint, users can set output settings using the -O -R and -S switches, and set the dataset or mart.dataset using the -d switch.
   c. -E commandline switch : By far, the most flexible way to batchMQL queries against one or more Marts and Datasets is to use MQL script files and the -E MartShell switch. These are text files with one or more valid MQL commands. Each MQL command must end with a semi-colon. MQL commands can be spread across multiple lines, with whitespace used to make more complex queries more readable. MQL scripts can contain many more individual commands than would be allowed on a single shell commandline. Also, MQLcommands can include quotes and other special characters that are difficult to include on a shell commandline, because of its pre-processing of the command. MQL script files can also contain comments. Any line beginning with the pound sign '#' is ignored by the MartShell processor. This allows users to write MQL scripts, which are easy to maintain over time. In addition

to the -E switch, users can use all other commandline switches except -e. Because there are MQL statement equivalents to all commandline switches, users are given the choice of which style best fits their needs. In some cases, a single script may be used in different contexts, depending on the commandline switches used in addition to the -E switch (this should be documented in the mql script), while in others, the entire procedure can be encapsulated and documented with MQL commands and comments within the script.

MartEditor

MartEditor is an administration tool designed for the generation, viewing and editing of the DatasetConfig XML files used to configure BioMart User Interfaces. Importing and exporting of the XML works from files and BioMart databases (where the DatasetConfig files are stored in a meta_configuration table). Users navigate through the configuration file using a convenient tree interface in the left panel, editing of the XML attributes for each node using the main right panel. Multiple files can be opened at once, and copying and pasting both between and within each DatasetConfig file is supported. Although MartEditor offers a convenient BioMart-specific editing experience, the main advantage over using a standard XML-editing tool are the extra features MartEditor offers in the creation and maintenance of the configuration files:

a) Hiding filters and attributes: Any of the XML objects can be set to 'hidden', meaning they are not loaded and do not appear in the BioMart interfaces. Any section of the XML file can be set to hidden with a simple right click and selecting the "toggle hidden" option.
b) Making filters drop down: Filters can be converted from a text-entry field to a drop down menu. Again this just involves a right-click on the corresponding FilterDescription and selecting the "make drop down" option. MartEditor then performs the SELECT DISTINCT column_name FROM table_name query, changes the FilterDescription type to "list" and inserts an option for each distinct result.
c) Automate_push_action: In situations where you want the choice of an option from one drop-down filter to affect the options in another filter push actions need to be set. For instance choosing chromosome 1 in the chromosome name menu should change the band options in the band start and end filters. To accomplish this you can right click on a FilterDescription with options set and select "automate push action". The dialog box then asks for the internal name of the filter to be set by the choice of this filter. In the above example this would be band_start or band_end. MartEditor then performs the query against the lookup table specified by this filter to set the appropiate push actions per option. There is an additional dialog box to add an "order by column_name" clause to the query if an alphabetical order is not what is required.
d) Creating an initial 'naïve' configuration of the dataset: The initial creation of a DatasetConfig file for a new BioMart dataset can be a long painstaking process if the whole file is created from scratch. MartEditor offers a convenient shortcut to this by creating an initial guess at the required configuration file. An understanding of the BioMart system together with the database naming convention is used to produce XML capable of configuring functional MartExplorer and MartShell interfaces. In practise these need little further editing beyond copying and pasting to arrive at a final configuration. After selecting the naïve configuration option from the File menu and choosing one of the datasets available in the currently connected

database, the naïve XML generation starts. The basic logic is that each main table column should represent a filter, and each main or dimension table column should represent a potential attribute. In addition certain lookup table columns should form filters. The system cycles through each column in the main, dimension and lookup tables. Columns ending in _key (key columns used for joins in query generation) and columns only containing NULLs are ignored. In addition columns duplicated between main tables in datasets with a conformed dimension model are also ignored. Main table columns and lookup table columns starting glook_ or silent_ generate a new FilterDescription. These FilterDescriptions are organised into separate FilterCollections per table. Internal names and display names are generated from the column name or the content section of the table name for the FilterDescriptions and FilterCollections respectively. The exception to the generation of the filters is the id_list FilterCollection which ends up representing the external_identifier filters in the final interface. Main table columns ending _bool become boolean filters represented as individual options in the id_list_filters FilterDescription. Dimension table columns ending list become list filters in the id_list_limit_filters drop down FilterDescription. Main table columns not ending _bool and dimension table columns end up as an AttributeDescription and again these are organised into separate AttributeCollections per table. Usually the first stage in hand-finishing the XML generation after the naïve generation is to hide those filters and attributes that are not required. It is best to move these to separate 'hidden unwanted' filter and attribute pages. The benefit of this over straightforward deletion is that these filters and attributes will not reappear when the XML is updated in the future (see below). After this it is usually just a case of reorganising the Filter and AttributeDescriptions into more relevant Pages, Groups and Collections and editing some of the display and internal names. Finally some of the filters may require converting to drop down or having push actions added (see above).

e) Updating the configuration: Rather than having to remember all the database changes since the last time the XML was generated and adding these as new filters and attributes manually, MartEditor offers an automated update function. In the first stage of this all the Filter and AttributeDescriptions are checked against the new database to see if the table and column still exist. If not the Description is set to hidden. In the second stage the database is scanned for new filters and attributes. The logic used is the same as for the naïve XML generation. These appear in NEW_FILTERS and NEW_ATTRIBUTES pages, ready for copying and pasting into the existing XML. In addition any FilterDescriptions with drop-down menus are updated along with any enclosed push actions.

Appendix A

MartView URL-based batch querying examples

The basic url is http://www.ensembl.org/Multi/martview?stage=X where X can be **start**, **filter** or **output** followed by one or more cgi name=value pairs separated by &'s. The stages initialised are specified with **stage_initialised**=X where X is one of the three stages above. The species is specified by **species**=X (ie Homo_sapiens) and focus by **focus**=X where X can be 'gene', 'estgene', 'vega' or 'snp' for ensembl_mart. To export the final data replace **stage**=X with **_export**=1. The following examples demonstrate how to build up a URL query although usually only the final export one will be used for batched queries:

(a) initialise the start page to human ensembl genes

http://www.ensembl.org/Multi/martview?stage=start&stage_initialised=start&species=Homo_sapiens&focus=gene

(b) move to the filter stage for human ensembl genes

http://www.ensembl.org/Multi/martview?stage=filter&stage_initialised=start&stage_initialised=filter&species=Homo_sapiens&focus=gene

(c) add an ID list filter

http://www.ensembl.org/Multi/martview?stage=filter&stage_initialised=start&stage_initialised=filter&species=Homo_sapiens&focus=gene&named_gene=1&named_gene_filter=FG_AFFY-HG-U95Av2_ID&named_gene_list=32864_at,41214_at,31534_at,36367_at

(d) go to the output stage for this query. Options for **outtype** include 'attribute', 'snp_attribute', 'gene_structure' and 'sequence' for ensembl_mart. Options for **outformat** include 'html', 'txt', 'csv', 'tsv', 'xls', 'gff', 'fasta' and 'adf'.

http://www.ensembl.org/Multi/martview?stage=output&stage_initialised=start&stage_initialised=filter&stage_initialised=output&species=Homo_sapiens&focus=gene&named_gene=1&named_gene_filter=FG_AFFY-HG-U95Av2_ID&named_gene_list=32864_at,41214_at,31534_at,36367_at&outtype=attribute&outformat=html

(e) choose some attributes

http://www.ensembl.org/Multi/martview?stage=output&stage_initialised=start&stage_initialised=filter&stage_initialised=output&species=Homo_sapiens&focus=gene&named_gene=1&named_gene_filter=FG_AFFY-HG-U95Av2_ID&named_gene_list=32864_at,41214_at,31534_at,36367_at&outtype=attribute&outformat=html&attribute_list_gene=gene_name&attribute_list_affy=1&attribute_list_affy_this=xaffy_hg_u95av2_dis

(f) finally export this data

http://www.ensembl.org/Multi/martview?_export=1&stage_initialised=start&stage_initialised=filter&stage_initialised=output&species=Homo_sapiens&focus=gene&named_gene=1&named_gene_filter=FG_AFFY-HG-U95Av2_ID&named_gene_list=32864_at,41214_at,31534_at,36367_at&outtype=attribute&outformat=html&attribute_list_gene=gene_name&attribute_list_affy=1&attribute_list_affy_this=xaffy_hg_u95av2_dis

(g) Get some sequence for these genes (Note that the gene_name attribute is required for sequence fetching).

Available options for **seq_scope** are 'gene' and 'tscr'. Options for **seq_scope_type** include 'gene_only', 'gene_5_3', 'gene_5', 'gene_3',

'5_only', '3_only', 'exons', 'exons_5_3', 'exons_5', 'exons_3', 'cdna', 'coding', 'coding_translation', 'upstream_utr', 'upstream_utr_5', 'downstream_utr', 'downstream_utr_3' for transcripts(tscr) and 'gene_only', 'gene_5_3', 'gene_5', 'gene_3', '5_only', '3_only', 'exons', 'exons_5_3', 'exons_5', 'exons_3' for genes (gene). Flanking sequences are defined with ‘seqscopetscr_5_flank’ and ‘seqscopetscr_3_flank’.

http://www.ensembl.org/Multi/martview?_export=1&stage_initialised=start&stage_initialised=filter&stage_initialised=output&species=Homo_sapiens&focus=gene&named_gene=1&named_gene_filter=FG_AFFY-HG-U95Av2_ID&named_gene_list=32864_at,41214_at,31534_at,36367_at&outtype=sequence&outformat=html&attribute_list_gene=gene_name&seq_scope=tscr&seq_scope_type=coding_translation&seq_scopetscr_5_flank=10

CGI names for filters and attributes

The place to find the cgi names to add to the URL is in the MetaData.pm module of the web installation. The checkboxes on the web page have a cgi_name corresponding to $form_name in most cases:

```
in_encode,
transcript_count,
utr5_available,
utr3_available,
human__paralogss,
human__homologss,
chimp__homologss,
chicken__homologss,
mouse__homologss,
rat__homologss,
fugu__homologss,
danio__homologss,
drosophila__homologss,
mosquito__homologss,
c_elegans__homologss,
c_briggsae__homologss,
human_upstream_regions,
mouse_upstream_regions,
rat_upstream_regions,
chicken_upstream_regions,
c_elegans_upstream_regions,
c_briggsae_upstream_regions,
syn_snp,
attribute_list_chr,
attribute_list_chr,
attribute_list_protein,
attribute_list_protein_interpro,
attribute_list_protein_clustr,
attribute_list_protein_go,
attribute_list_protein_other,
attribute_list_gene,
attribute_list_gene,
attribute_list_xref,
attribute_list_affy,
attribute_list_disease,
attribute_list_genesnp,
expression_attribs
attribut_list_interpro,
attribut_list_family,
attribute_list_domain,
attribute_list_snp,
attribute_list,
attribute_list_freq,
attribute_list_snp,
attribute_list_snp_loc,
attribute_list_snp_gene_related_gene,
attribute_list_snp_gene_related_snp,
attribute_list_snp_gene_related_snp,
attribute_list_snp_gene_related_snp,
attribute_list,
attribute_structure,
```

The cgi_names of the other filters are defined using the methods set_cgi_name or set_name_suffix which adds a suffix to the existing form name:

```
$entry->set_cgi_name('focus');
$entry->set_cgi_name('species');
#$entry->set_cgi_name('chr_check');
$entry->set_cgi_name('chromosome_name');
#$entry->set_cgi_name('mapweight_limit');
#$entry->set_cgi_name('chr_check');
$entry->set_cgi_name('chromosome_name');
#$entry->set_cgi_name('qtl_check');
$entry->set_cgi_name('trait_name');
$entry->set_cgi_name('trait_region');
$entry->set_cgi_name('trait_last_region');
$entry->set_cgi_name('hiddenchrstart');
$entry->set_cgi_name('hiddenchrend');
$entry->set_cgi_name('hiddenchrname');
#$entry->set_cgi_name('encode_check');
$entry->set_cgi_name('type_name');
$entry->set_cgi_name('encode_region');
$entry->set_cgi_name('encode_last_region');
$entry->set_cgi_name('hiddenchrstart');
$entry->set_cgi_name('hiddenchrend');
$entry->set_cgi_name('hiddenchrname');
$entry->set_cgi_name('named_protein_file');
$entry->set_cgi_name('prot_mol_function');
$entry->set_cgi_name('mol_button');
$entry->set_cgi_name('prot_biol_process');
$entry->set_cgi_name('biol_button');
$entry->set_cgi_name('prot_cell_location');
$entry->set_cgi_name('cellular_button');
$entry->set_cgi_name('evidence_prot_mol');
$entry->set_cgi_name('evidence_prot_biol');
$entry->set_cgi_name('evidence_prot_comp');
$entry->set_cgi_name("_named_gene_file");
$entry->set_cgi_name('named_gene_filtertype');
$entry->set_cgi_name('named_gene_file');
$entry->set_cgi_name("_named_fam_file");
$entry->set_cgi_name('named_fam_file');
$entry->set_cgi_name("_named_snp_file");
$entry->set_cgi_name('named_snp_file');
$entry->set_cgi_name('strain_name');
$entry->set_cgi_name('population_name');
$entry->set_cgi_name('freq_limit');
$entry->set_cgi_name('population_name2');
$entry->set_cgi_name('max_freq_limit');
$entry->set_cgi_name('max_population_name2');
$entry->set_cgi_name('ratio');
$entry->set_cgi_name('snp_broad');
$entry->set_cgi_name('snp_narrow_hidden');
$entry->set_cgi_name('ratio');
$entry->set_cgi_name('strain_name');
$entry->set_cgi_name('expression_type');
$entry->set_cgi_name('seq_scope_img');
$entry->set_name_suffix(‘type’);
```

Some filters such as **named_gene_filter** take a list of filters defined in one of the modules (ie MartGeneExtractor.pm) starting with a given prefix such as FG_.

Appendix B

MartShell query examples

Interactive

```
# List mart databases
list marts;

# List all available datasets
list datasets all;

# List datasets for a given mart
list datasets Ensembl

# or
set mart Ensembl;
list datasets;

# sets dataset;
use Ensembl.hsapiens_gene_ensembl;

# or, if mart is set
use hsapiens_gene_ensembl;

# now list availble attributes and filters;
list attributes;
list filters;

# <tab><tab> will do the same thing
# eg. get <tab><tab> where <tab><tab>

# now a real query
using MSD.msd get pdb_id where resolution_less < 0.9;

# chain examples
using MSD.msd get pdb_id where resolution_less < 1.5 and has_ec_info only as q;
using Ensembl.hsapiens_gene_ensembl get sequence transcript_flanks+1000
where pdb in q;

using Ensembl.hsapiens_gene_ensembl get uniprot_sptrembl where
disease_gene_boolean only and chr_name=22 as q;
using Uniprot.uniprot get protein_sequence where sprot_id in q;

#count focus objects (genes) for  above query
count_focus_from
using Ensembl.hsapiens_gene_ensembl get uniprot_sptrembl where
disease_gene_boolean only and chr_name=22 as q;
using Uniprot.uniprot get protein_sequence where sprot_id in q;

using Ensembl.hsapiens_gene_ensembl get uniprot_sptrembl where has_nonsynsnp only and
disease_gene_boolean only as q;
using Uniprot.uniprot get sptr_ac ,protein_name where sprot_id in q;

# named procedures
using Ensembl.hsapiens_gene_ensembl get uniprot_sptrembl where disease_gene_boolean only as diseaseGenes;
execute procedure diseaseGenes;


using ? get pdb_id where resolution_less < ? as pdbSet;
execute procedure pdbSet(MSD.msd,0.9);
count_focus_from pdbSet(MSD.msd,0.9);

using Ensembl.hsapiens_gene_ensembl get sequence transcript_flanks+1000 where pdb in pdbSet(MSD.msd,0.9);

# outputting to a file
set output file='myfile';
using Ensembl.hsapiens_gene_ensembl get gene_stable_id;

# back to interactive mode
unset output file;
```

```
#set the mart and dataset, then get counts
set Mart MSD;
use msd;
count_focus_from get pdb_id where resolution_less < 0.9;

#count the number of focus objects in a particular dataset
count_focus_from Ensembl.hsapiens_gene_ensembl;

set Mart Ensembl;
count_focus_from hsapiens_gene_ensembl;
```

Batch

1. Simple example. Default Output (tab separated, sent to STDOUT) is used. All Mart connection parameters and DatasetConfiguration locations are specified in $HOME/.martj_adaptors.xml. Mart and Dataset are set inside the MQL:

```
myShell>martshell.sh -e 'using Ensembl.hsapiens_gene_ensembl get gene_stable_id'
ENSG00000187981
ENSG00000187908
ENSG00000188553
ENSG00000129899
ENSG00000184895
ENSG00000129824
ENSG00000067646
ENSG00000176679
ENSG00000099715
ENSG00000173394
ENSG00000168757
.
.
.
```

2. Redirect output to gene_info.out, separating fields by commas

```
myShell>martshell.sh -O gene_info.out -F tabulated -S',' -e 'using Ensembl.hsapiens_gene_ensembl
get gene_stable_id, transcript_stable_id'
myShell>ls -l gene_info.out
-rw-r--r--    1 dlondon  ensembl     32000 Jun  3 11:07 gene_info.out
myShell>head -2 gene_info.out
ENSG00000187981,ENST00000343885
ENSG00000187908,ENST00000339871
```

3. Get an output of peptides for disease genes on chr 22 in Fasta format, sent to pepout.fa

```
myShell>martshell.sh -Opepout.fa -F fasta -e 'using Ensembl.hsapiens_gene_ensembl get sequence
peptide where chr_name = 22 and disease_gene_boolean only'
myShell>ls -l pepout.fa
-rw-r--r--    1 dlondon  ensembl     35357 Jun  3 11:13 pepout.fa
myShell>head -10 pepout.fa
>ENST00000334029.1|ENSG00000100033.4|ENSP00000334726.1 strand=revearse|chr=22|assembly=NCBI34|peptide sequence
MALRRALPALRPCIPRFVPLSTAPASREQPAAGPAAVPGGGSATAVRPPVPAVDFGNAQEAYRSRRTWELARSLLVLRLC
AWPALLARHEQLLYVSRKLLGQRLFNKLMKMTFYGHFVAGEDQESIQPLLRHYRAFGVSAILDYGVEEDLSPEEAEHKEM
ESCTSAAERDGSGTNKRDKQYQAHWAFGDRRNGVISARTYFYANEAKCDSHMETFLRCIEASGRVSDDGFIAIKLTALGR
PQFLLQFSEVLAKWRCFFHQMAVEQGQAGLAAMDTKLEVAVLQESVAKLGIASRAEIEDWFTAETLGVSGTMDLLDWSSL
IDSRTKLSKHLVVPNAQTGQLEPLLSRFTEEEELQMTRMLQRMDVLAKKATEMGVRLMVDAEQTYFQPAISRLTLEMQRK
FNVEKPLIFNTYQCYLKDAYDNVTLDVELARREGWCFGAKLVRGAYLAQERARAAEIGYEDPINPTYEATNAMYHRCLDY
VLEELKHNAKAKVMVASHNEDTVRFALRRMEELGLHPADHRVYFGQLLGMCDQISFPLGQAGYPVYKYVPYGPVMEVLPY
LSRRALENSSLMKGTHRERQLLWLELLRRLRTGNLFHRPA*
.
.
```

4. Specify the Mart and Dataset using the -d switch

```
myShell>martshell.sh -d Ensembl.hsapiens_gene_ensembl -e'get gene_stable_id'
ENSG00000187981
ENSG00000187908
ENSG00000188553
```

ENSG00000129899
ENSG00000184895
ENSG00000129824
ENSG00000067646
ENSG00000176679
ENSG00000099715
ENSG00000173394
.
.
.

5. Specify an alternative MartRegistry file with Mart Connection parameters, and DatasetConfiguration locations.

myShell>martshell.sh -R ~/myOtherAdaptors.xml -e'using myMart.hsapiens_gene_ensembl \
get gene_stable_id where disease_gene_boolean excluded'
ENSG00000187981
ENSG00000187908
ENSG00000188553
ENSG00000129824
ENSG00000067646
ENSG00000176679
ENSG00000099715
ENSG00000173394
ENSG00000168757
ENSG00000183335
ENSG00000182323
ENSG00000186406
.
.
.

6. Pipe the output of a mart query into another command (*NIX only)

myShell>martshell.sh -e 'using Ensembl.hsapiens_gene_ensembl \
get gene_stable_id, hugo, sanbi_expression \
where chr_name = 2 and marker_start = D2S2216 and \
marker_end = D2S2159 and known_gene_boolean only and \
hugo_boolean only' | grep 'ANAT SITE.*pancreas.*CELL TYPE.*Langerhans'
ENSG00000172071 EIF2AK3 ANAT SITE:Alimentary -> pancreas DEV STAGE:Adult PATH:Normal CELL TYPE:Langerhans cell PREP:bulk

7. Embed calls to MartShell inside other applications. Here is an example perl script that calls for all human gene ids with gene ids of chimp, rat and mouse orthologs from Ensembl, and uses these ids within the application for other uses.

```
#!/usr/bin/perl -w

use Bio::SeqIO;

#this requires that a java 1.4.* vm be on the path, along with the martj/bin directory
#also requires that either $MARTJHOME/data/defaultRegisty.xml, or $HOME/.martj_adaptors.xml
#contains connection to mart at ebi or ensembl

#the order of these matches the order of the attributes in $baseMQL
my @dataset = qw(Ensembl.hsapiens_gene_ensembl
                 Ensembl.mmusculus_gene_ensembl
                 Ensembl.ptroglodytes_gene_ensembl
                 Ensembl.frubripes_gene_ensembl
                 Ensembl.rnorvegicus_gene_ensembl
                );

#get human id, and required orthologs in the order that the perl script
#expects them, using a file of interpro ids as the list filter
#this file can be changed each time the script is run to
#get different ortholog lists
$baseMQL = qq(using Ensembl.hsapiens_gene_ensembl get gene_stable_id, mouse_ensembl_id ,
chimp_ensembl_id , fugu_ensembl_id, rat_ensembl_id where interpro_ids in myInterproIds.txt);

my $baseMQLCommand = "martshell.sh -F tabulated -S',' -e '$baseMQL'";
```

```
#print "$baseMQLCommand\n";
#exit;

open (BASEMQLOUTPUT, "$baseMQLCommand |") or die "Couldnt run baseMQL $!\n";

my %humanIds = ();

while (<BASEMQLOUTPUT>){
  chomp;
  my @ids = split /(,)/; #keep commas, and undefined fields

  my $humanId = shift @ids;

  my $iter = 1;
  foreach my $orthoId (@ids) {
    next if ($orthoId eq ',');

    #skip cases where there is no ortholog id between human and this species
    if ($orthoId) {
      $orthoId =~ s/\.\d+$//;
      my $dataset = $dataset[$iter];
      $humanIds{$humanId}->{$dataset}->{$orthoId} = 1; #unique list
    }
    $iter++;
  }
}
close BASEMQLOUTPUT;

my $execTemplate = 'martshell.sh -F fasta -E %s';
my $mqlFileTemplate = '%s.mql';
my $outputFileTemplate = '/scratch/out/%s.fa';

my $mqlTemplate = "using %s get sequence 1000+gene_exon_intron where gene_stable_id = %s;\n";

foreach my $humanId (keys %humanIds) {
  my $mqlFile = sprintf($mqlFileTemplate, $humanId);

  open(MQLFILE, ">$mqlFile");
  printf MQLFILE ($mqlTemplate, $dataset[0], $humanId);

  foreach my $dataset(keys %{$humanIds{$humanId}}) {
    foreach my $orthoId (keys %{$humanIds{$humanId}->{$dataset}}) {
      printf MQLFILE ($mqlTemplate, $dataset, $orthoId);
    }
  }
  close MQLFILE;

  my $outputFile = sprintf($outputFileTemplate, $humanId);
  my $execCommand = sprintf($execTemplate, $mqlFile);

  open (EXECMQL, "$execCommand |") or die ("couldnt exec $execCommand: $!\n");

  my $mqlInput = \*EXECMQL; #glob the handle and pipe it through Bio::SeqIO

  my $seqIn = Bio::SeqIO->new(q(-fh) => $mqlInput, q(-format) => q(fasta)) or die ("couldnt process with BioPerl
$!\n");
  my $seqOut = Bio::SeqIO->new(q(-file) => ">$outputFile", q(-format) => q(fasta));

  while (my $seq = $seqIn->next_seq) {
    $seqOut->write_seq($seq);
  }
  close EXECMQL;
}

exit;
```

Appendix C

Glossary of Terms

1. Mart: Represents a source of one or more Mart Datasets. Currently only connections to RDBMS systems are supported, but in the future this concept will be extended to also support connections to web services, which wrap one or more RDBMS systems hosting Marts.
2. Dataset: A single source of data, which includes various attributes, and filtering criteria associated with a focus object or set of focus objects (gene/transcript, snp, etc.).
3. Attribute: Some feature of a focus object made available by a Mart Dataset. Most attributes are precomputed during the production of the dataset to increase performance, but some are dynamically calculated.
4. Filter: A feature of a focus object that can be used to filter the objects returned in a query. In most cases, filters are linked with corresponding attributes, e.g., you can filter for objects meeting some criteria and get the value of that criteria in the attribute output.
5. DatasetConfiguration: Configures a Mart user interface to allow queries against a dataset. Describes the attributes and filters that a dataset makes available to the user interface, and instructs it on how it should present these to the user. Configurations can also limit the kinds of queries users can ask, by restricting the number or combination of certain attributes that can be combined in a single query. Each dataset can have more than one configuration describing different combinations of attributes and filters. Typically, all datasets are deployed with a default configuration, and for most datasets, this is the only configuration made available. Configurations can be stored as XML files on the file system, or stored within a Mart (e.g. ultimately, in an RDBMS database table). Mart user interface applications automatically load any configurations that a mart contains when the user connects to that mart, and also allow users to load individual configurations from the file system manually.
6. MartRegistry: Describes the location of one or more Marts (eg. Connection settings), other MartRegistry files, or individual Configurations to the user interface. Mart user interfaces are able to load and save MartRegistry files to the file systems like bookmarks.

Appendix D

Overview of BioMart User Interface Operations

A. Operations common to all interfaces:
   1. Set the dataset to use for a query
   2. Choose filtering criteria
   3. Choose attributes
   4. Choose output options (file, format, separator, etc.)
   5. Execute query

B. Optional Operations available to Java Mart User Interface Applications
   1. Set the Mart to use for a query
   2. Connect to a Mart this option creates a connection to the service, determines which datasets it hosts, and automatically loads any dataset configurations it makes available, associating each configuration with the Mart and its specific Dataset.
   3. Load a Dataset Configuration file from the file system this option associates the configuration with its intended dataset, but does not automatically connect to any Marts, nor does it associate the configuration with any Marts to which the user has already connected (there are ways the user can manually specify these associations when using the configuration to query an existing Mart dataset).
   4. Load a MartRegistry file if the Registry contains information for connecting to an existing Mart, then all of the behavior associated with connecting to a Mart (option 1) occurs. If it contains information for loading a Dataset Configuration file, then all the behavior associated with loading this Configuration File (option 2) occurs. If it contains information for loading another MartRegistry, then the system recursively harvests this Registry and any referring Registry locations, for Mart connections, or Dataset Configuration file locations.
   5. Specify an alternate configuration for a particular Mart and dataset to be queried
   6. Save queries for later reuse
   7. Load a saved query, or set of queries
   8. Use the results of one single-attribute query as the values in a filter for a second query (query chaining)