

BioMart 0.7 Documentation

Table of Contents

1 Introduction.....	3	5 Configuring Marts.....	47
1.1 Contacts.....	3	5.1 Server-side Configuration.....	47
1.2 Description.....	3	5.1.1 Dataset configuration file.....	47
1.3 Licence.....	3	5.1.2 MartEditor.....	47
1.4 System Overview.....	3	5.1.3 Upgrading 0.5 to 0.6/0.7.....	53
2 Installing BioMart.....	5	5.2 Client-side Configuration.....	53
2.1 Downloading martj.....	5	5.2.1 Registry Files.....	53
2.1.1 Binary distribution.....	5	6 Querying Marts.....	56
2.1.2 Source distribution.....	5	6.1 MartShell.....	56
2.2 Downloading biomart-perl.....	6	6.1.1 Starting.....	56
2.3 Installing martj.....	6	6.1.2 Using.....	56
2.4 Installing biomart-perl.....	6	6.1.3 Batch jobs and scripting.....	57
2.4.1 Prerequisites.....	6	6.1.4 MQL guide.....	58
2.4.2 Setting the Registry.....	10	6.2 MartExplorer.....	61
2.4.3 Configuring.....	10	6.2.1 Starting.....	61
2.4.4 Starting and stopping MartView.....	12	6.2.2 Building queries.....	62
2.4.5 Troubleshooting MartView startup.....	12	6.3 MartView.....	63
2.4.6 MartView maintenance tasks.....	13	6.3.1 Web browser interface.....	63
3 QuickStart Guide.....	15	6.3.2 Web services interface.....	64
3.1 Creating a mart.....	15	6.4 Perl API.....	64
3.1.1 Handmade marts.....	15	6.4.1 Prerequisites.....	64
3.1.2 Auto-generated marts.....	16	6.4.2 Examples.....	65
3.2 Configuration using MartEditor.....	21	6.4.3 Extra functions.....	67
3.3 Setting up MartView.....	22	6.5 Java API.....	67
4 Building Marts.....	24	6.5.1 Prerequisites.....	68
4.1 Structure of a Mart.....	24	6.5.2 Examples.....	68
4.2 Data model.....	25	6.6 Web Services API - RESTful Access.....	70
4.3 MartBuilder.....	26	6.6.1 Metadata.....	70
4.3.1 Starting.....	26	6.6.2 Queries.....	72
4.3.2 Schema editor vs. Dataset editor.....	26	6.7 Web Services API - SOAP Access.....	73
4.3.3 Connecting to a source database.....	26	6.7.1 Semantic Annotations (WSDL-S).....	74
4.3.4 Multiple source schemas (Partitioned Schema).....	27	6.8 MartView URL Requests.....	74
4.3.5 Adjusting the schema.....	28	6.9 MartView XML Requests.....	75
4.3.6 Creating a dataset.....	30	6.10 DAS Server.....	75
4.3.7 Adjusting the dataset.....	30	7 Third-party Software.....	78
4.3.8 Generating SQL.....	35	7.1 BiomaRt.....	78
4.3.9 Saving your work.....	36	7.2 Taverna.....	79
4.3.10 MartBuilder options in full.....	36	7.3 Galaxy.....	79
4.4 MartRunner.....	42	7.4 Ensembl.....	79
4.4.1 Starting and stopping MartRunner.....	43	7.5 GMOD.....	79
4.4.2 Sending jobs to MartRunner.....	43	7.6 Bioclipse.....	79
4.4.3 Monitoring MartRunner.....	43	7.7 WebLab.....	80
4.5 Configuring MartBuilder (optional).....	46		

1 Introduction

1.1 Contacts

The BioMart project website can be found at <http://www.biomart.org/> where the latest version of the BioMart software can be downloaded and this documentation and other related information can be found.

If you have any questions about setting up or using BioMart software, please send them by email to mart-dev@ebi.ac.uk.

The BioMart project is developed jointly by the European Bioinformatics Institute (<http://www.ebi.ac.uk/>) and the Cold Spring Harbor Laboratory (<http://www.cshl.edu/>).

1.2 Description

The purpose of BioMart is to convert one or more data source (flat files or relational) into data marts which can be accessed via its standardised web browser interface and also via its Perl, Java and webservice APIs.

The system comes with built-in support for query-optimization and database federation. BioMart software provides users with the ability to conduct fast, powerful queries using a selection of web, graphical and text based applications. Programmatic execution of queries is also available via a web-services API, or direct-access software libraries written in Perl and Java. For data providers, the system simplifies the task of integrating their own data with other datasets hosted on the network.

All the software is available for local installation.

1.3 Licence

BioMart software is completely Open Source, licensed under the LGPL, and freely available to anyone to use and redistribute without restriction. However, we would appreciate it if you credit its use should you choose to deploy or redistribute any part of it within your own project.

The full text of the LGPL licence is available at: <http://www.gnu.org/licenses/lgpl.html>.

1.4 System Overview

BioMart is designed around a three tier architecture.

The first tier consists of one or more relational databases. Each of these databases can hold one or more marts, which are schemas compliant with BioMart definitions. Inside each mart can be a number of individual datasets. Dataset configuration is stored in additional tables inside each mart and is created using MartEditor.

Two tools are provided to build and configure the mart databases in the first tier:

- MartBuilder, to construct SQL statements that will transform your schema into a mart.
- MartEditor, to configure the finished mart for use with the rest of the system.

The second tier contains two APIs - one written in Perl (distributed in the *biomart-perl* package) and the other in Java (distributed in the *martj* package).

The third tier consists of the query interfaces:

- MartView, a web browser interface, based on the Perl API.
- MartService, a web services interface, based on the Perl API.
- MartURLAccess, a URL based access to MartView, based on Perl API.
- MartExplorer, a standalone GUI tool, based on the Java API.
- MartShell, a command-line tool, also based on the Java API.

Dataset configuration is stored in XML format in a special table inside the database schema that the mart lives in.

A registry XML file on the client-side, managed by the user, dictates which datasets in which marts on which database servers are available for querying.

BioMart version 0.7 supports three major relational database platforms for hosting marts: MySQL, Oracle and Postgres.

2 Installing BioMart

The BioMart components are distributed in two separate packages.

biomart-perl contains the Perl API and all the BioMart applications that depend on it, such as MartView, MartService, MartURLAccess and DAS Annotation Server (if configured).

martj contains the Java API and all the BioMart applications that are written in Java, such as MartEditor, MartShell, MartExplorer and MartBuilder.

2.1 Downloading martj

2.1.1 Binary distribution

The best way to obtain *martj* is to download one of the pre-compiled binary distributions from <http://www.biomart.org/>.

The [martj-bin.zip](#) distribution is for Windows users, and contains the Java API and all the Java-based BioMart applications. The *bin* folder contains a number of *.bat* scripts for launching the applications. You can unpack it using WinZip or a similar application.

The [martj-bin.tgz](#) distribution is identical to the *martj-bin.zip* distribution, but is intended for Unix and Linux users. Instead of *.bat* scripts, it contains a number of *.sh* scripts which perform equivalent tasks. You can unpack it using:

```
tar -zxvf martj-bin.tgz
```

The [martj-bin.dmg](#) distribution is for MacOSX users, and contains bundles in the *bin* folder for each of the various Java-based BioMart applications. The *bin* folder also contains the MartShell application *.command* script, and the *lib* and *data* folders contain the files that MartShell depends on. The *.dmg* image file will automatically unpack itself when double-clicked.

2.1.2 Source distribution

The source code for *martj* is available for download via CVS, but you will need the ant tool installed if you subsequently wish to compile it. ant is available from <http://ant.apache.org/>. Download and install it as per the instructions on that website.

To check *martj* out from CVS, you need to type the following commands on the command line prompt. The password you need to enter when prompted is **CVSUSER**.

```
cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/biomart login
cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/biomart \
co -r release-0_7 martj
```

To compile *martj* using *ant*, change into the *martj* directory created by the commands above and type:

```
ant jar
```

This will create the *martj.jar* file inside the *build* folder. All other JAR files which the Java-based BioMart applications depend on can be found in the *lib* folder.

2.2 Downloading biomart-perl

There is no binary distribution for *biomart-perl*. It is only available as a source distribution from CVS.

To check *biomart-perl* out from CVS, you need to type the following commands on the command line prompt. The password you need to enter when asked is **CVSUSER**.

```
cvcs -d :pserver:cvcsuser@cvcs.sanger.ac.uk:/cvsroot/biomart login
cvcs -d :pserver:cvcsuser@cvcs.sanger.ac.uk:/cvsroot/biomart \
  co -r release-0_7 biomart-perl
```

No compilation is required, but it will need to be configured before you can use it. For details on how to configure it, please refer to the section on installing *biomart-perl*.

2.3 Installing martj

You need to have Java 1.3 or later installed. You can get Java from <http://java.sun.com/>.

martj has been tested with Java 1.3, 1.4 and 1.5.

If you attempt to run any *martj* component on Java 1.6 or higher, and experience problems, please try the same series of actions with Java 1.5 before reporting a bug.

martj requires no further installation, except if you wish to use marts other than the ones defined by default. To do so requires you to modify the Registry file, which is discussed elsewhere in this document.

It is important that you do not modify the directory structure inside the *martj* folder, or move or copy any of the scripts from the *bin* folders to other locations. Running these scripts from other locations will not work.

2.4 Installing biomart-perl

2.4.1 Prerequisites

You need to have Perl version 5.6.0 or later installed first. You can get the latest version of Perl from <http://www.perl.org/>.

It is important that you do not modify the directory structure inside the *biomart-perl* folder, or move or copy any of the files within it to other locations.

biomart-perl depends on a number of Perl modules. When you run the configuration steps detailed elsewhere in this document it will tell you if any of the ones it needs are missing so

the best plan is to run the `configure.pl` straight away from your *biomart-perl* directory and install any missing modules:

```
perl bin/configure.pl -r conf/registryURLPointer.xml
```

The easiest way to install these missing modules is to use CPAN shell. Unless you are knowledgeable about CPAN and know how to do otherwise, modules should be installed by the root user on Linux/Unix systems, or by the Administrator on MacOSX/Windows systems.

For each module reported as missing by the configuration step, type:

```
cpan -i Module::Name
```

You should replace **Module::Name** with the name of the module you are attempting to install, ideally by cut and pasting from the output from `configure.pl`. Read the questions CPAN asks during installation thoroughly, and answer **yes** when it asks you if you want to install missing dependencies. It is usually fine to accept the default responses for almost all questions it asks.

For reference, the list of required CPAN modules required at the time of writing is shown below. Version numbers are those that have been tested against, but other more recent versions may also work.

Dependency	Module name	Module version
API	XML-DOM	1.44
API	OLE-Storage_Lite	0.14
API	Exception-Class	1.23
API	libwww-perl	5.8
API	Log-Log4perl	1.05
API	Test-Exception	0.24
API	DBI and relevant DBD drivers	1.53
API	Digest::SHA	5.44
Website	IO::Compress::Gzip	2
Website	Number-Format	1.51
Website	Template-Toolkit	2.14
Website	Template-Plugin-Number-Format	1.01
Website	CGI-Session	4.14
Website	Readonly	1.13
Website	List-MoreUtils	0.22
Website	SpreadSheet-WriteExcel	2.17
Website	IO-Compress-Zlib	2.003
Website	SOAP-Lite	0.710.08

2.4.1.1 Apache installation for MartView

All references to MartView in this document that relate to configuration and maintenance equally apply to and affect MartService, as the two are part of the same application.

If you are going to be running a MartView/MartService/DAS server, you will also need to have an Apache web-server installed. This can be downloaded from <http://httpd.apache.org/>, and should be installed as per the guidelines on that website. You do not need to configure Apache to be used with BioMart, as the BioMart configuration scripts will handle that for you.

MartView works fine with all versions of Apache 1.3 or higher, including Apache 2.0 or higher.

MartView requires a few Apache extension modules to be installed. It does not matter if they are compiled into Apache or provided as dynamic modules. If you are missing any of them, the website where you can download them is listed beside each one.

Apache version	Module name	Module website
1.3, 1.4	mod_gzip <i>(optional, improves performance)</i>	http://sourceforge.net/projects/mod-gzip/
	mod_perl	http://perl.apache.org/
2.0 or higher	mod_deflate	Part of the Apache distribution.
	mod_perl	http://perl.apache.org/

MartView has been designed to work best with Apache 2.0 or higher but Apache 2.0 is not a prerequisite.

It is highly recommended that you install the appropriate compression module for Apache before running MartView. If you do not, then MartView is likely to be very slow.

For Apache 2.0 or higher, use *mod_deflate*. For Apache versions before 2.0, use *mod_gzip*. Details are in the table above.

The Perl API and MartView configuration scripts check which Apache modules are available by using *apxs* (Apache 1.3/1.4) or *apxs2* (Apache 2.0+). It requires this tool to live in the same location as the Apache binary (usually called *apache*, *apache2*, or *httpd*). If you have installed a binary distribution of Apache, you may also need to install the Apache development tools to make *apxs/apxs2* available.

Other Apache modules are also used by MartView but these are all available with the default Apache installation and so you should not need to worry about having to install them.

2.4.1.2 Apache and ModPerl quick setup

If you do not already have Apache and ModPerl installed on your system then you can follow these steps to set them up.

These steps assume a Unix/Linux-based system. For other operating systems, please refer to the Apache and ModPerl websites for instructions.

For the purposes of these instructions it is assumed that you will be installing the latest versions of Apache and ModPerl that were available at the time of writing (Apache 2.2.14 and ModPerl 2.0.4).

BioMart software does not depend on specific versions of Apache or ModPerl. It can use other versions if required but these instructions are only valid for the versions specified.

First you will need to create a directory where you can work. In our example we will install Apache in **/home/biomart/apache**.

```
mkdir /home/biomart/apache
```


You will need to substitute your directory for this example location in all the commands and explanations in this section.

Next you need to download, unpack, and build Apache inside this directory.

```
cd /home/biomart/apache
mkdir source
cd source
wget http://www.apache.org/dist/httpd/httpd-2.2.14.tar.gz
tar zxvf httpd-2.2.14.tar.gz
cd httpd-2.2.14
./configure \
    --enable-deflate \
    --prefix=/home/biomart/apache \
make install
```

Apache has now been built and configured. Your Apache installation path is `/home/biomart/apache/bin`. You will need this when configuring *biomart-perl* to use this copy of Apache.

The last step is to download and install ModPerl.

Note for ModPerl you may need to upgrade your Perl CGI module to the latest version (the module name is `CGI`). You can use the same technique to upgrade this as you used to install the other Perl module dependencies for *biomart-perl*.

This only applies when using ModPerl 2.0 or higher, as per this example. You will know if you need to upgrade if errors show in the Apache error log that refer to Apache/Response.

```
cd /home/biomart/apache/source
wget http://perl.apache.org/dist/mod_perl-2.0-current.tar.gz
tar zxvf mod_perl-2.0-current.tar.gz
cd mod_perl-2.0.4
perl Makefile.PL \
    PREFIX=/home/biomart/apache \
    MP_APXS=/home/biomart/apache/bin/apxs
make install
```

Now ModPerl has been installed, the setup of Apache and ModPerl is complete.

2.4.2 Setting the Registry

The BioMart Perl API and MartView allow users to make queries against a predetermined list of marts, defined in a Registry file. When using the Perl API, this registry file can be located anywhere the user requires.

However, when using MartView, the registry file to be used must be located in the *conf* folder of the *biomart-perl* installation.

Registry files are in XML format. You will find a number of example registry files already in the *conf* folder after you download and unpack *biomart-perl*. They can be extended to widen the selection of marts to include others available publicly, or they can be adapted to serve your own local marts.

The structure of the registry file is discussed elsewhere in this document.

2.4.3 Configuring

2.4.3.1 Configuring the BioMart Perl API

Configuration of the Perl API requires a single step. Change into the *biomart-perl* directory, then type:

```
perl bin/configure.pl -r conf/registryURLPointer.xml
```

where **registryURLPointer.xml** is the registry file you wish to use from the **conf** folder.

The first question the configure script will ask is:

```
Do you want to install in API only mode [y/n] [n]:
```

Type **y** to install the API only.

During configuration it may point out that required Perl modules are missing. If this happens, follow the steps detailed in the prerequisites section above to install these missing Perl modules.

When it has completed successfully, you will see this final message:

```
Looks good.... you are done.
```

2.4.3.2 Configuring MartView

This section describes how to set up Apache for use with MartView.

You will need a Registry file defined – the default one is in the **conf** folder and is called **registryURLPointer.xml**. See section 3.3 for details on how to create your own registry file.

MartView can use only one single Registry file at a time.

Before running the configuration script for MartView some settings need to be defined in the **settings.conf** file in the **conf** directory:

- **apacheBinary** – you should set the path to your apache httpd binary.
- **serverHost** – the hostname of the server to include in the apache configuration. Leaving it as localhost is fine for most cases.
- **port** – the port MartView should listen on for requests. The default one should work fine. If MartView will be the only application served on this machine you can change it to 80 so users do not have to enter a port number when communicating with MartView via a web browser.
- **proxy** – If your server is receiving port forwarded requests from a server other than the one it is running on then you should enter this server hostname here. As MartView needs to encode the hostname in some responses in order to redirect future requests, it needs to be told the hostname of the machine that the port-forwarded connections are coming from. For the normal scenario of no forwarding just leave blank.
- **location** – this setting affects the URL that will be used to access MartView. The URL will be formed of the server name, followed by the response to this question, followed by

the script name required. The default setting on biomart will be used in all URL examples in this documentation.

Other optional settings can be configured from within **settings.conf**, allowing you to specify amongst others the colour schemes and wording to use on the site, and to specify an alternative web server to use for relative URLs from query results. It is possible to enable background result jobs where the results are stored in a server-side directory and the user emailed when they are ready. The directory settings and mail options are all configured here as well. Other configurable options include how long session related data is stored on the server and how webservice logging is managed.

You may also like to edit the **site_header.tt** document in the **conf/templates/default** directory in order to embed the MartView interface into a custom setting, for instance by adding the logo and navigation bars from your website. Instructions for both **settings.conf** and **site_header.tt** are embedded within those files.

When you are done customising the settings, the same script is used to configure MartView as for the Perl API.

From the *biomart-perl* directory type

```
perl bin/configure.pl -r conf/registryURLPointer.xml
```

changing **registryURLPointer.xml** to the registry file in the **conf** directory you wish to use.

It will ask:

```
Do you want to install in API only mode [y/n] [n]:
```

Type **n** to install the Perl API and MartView together.

During configuration it may point out that required Perl modules are missing. If this happens, follow the steps detailed in the prerequisites section above to install these missing Perl modules.

MartView will now proceed to process the Registry file and download the various dataset configurations defined therein. It will build templates for then compile all the pages of the MartView website. This may take quite some time. The final message you see before completion should be:

```
Compiling templates for visible datasets
```

2.4.4 Starting and stopping MartView

Change to the *biomart-perl* directory and type (substituting **/my/chosen/Apache/binary** for the correct Apache location chosen during configuration):

```
/my/chosen/Apache/binary -d $PWD -f $PWD/conf/httpd.conf
```

Test MartView by pointing your web browser to the following URL, substituting **<host>**, **<port>** and **<location>** for the values you configured earlier:

```
http://<host>:<port>/<location>/martview
e. g.
http://localhost:5555/biomart/martview
```

To stop it again, change to the *biomart-perl* directory and type:

```
kill `cat logs/httpd.pid`
```

If the `httpd.pid` file is damaged or missing, you will have to identify and kill the Apache processes manually. This is potentially dangerous if you have more than one Apache instance running on your machine and so should be done with care. Therefore you should always be careful not to damage `httpd.pid`.

2.4.5 Troubleshooting MartView startup

The following information may help if you find that MartView will not start up correctly. Thanks to Eric Ross for providing most of it.

2.4.5.1 Useless use of AllowOverride (in console)

This is an Apache version issue and can be safely resolved by deleting the offending lines from `httpd.conf`.

2.4.5.2 Couldn't determine Username (in console)

Your operating system may not have a definition set up for the Apache web browser process owner. It can be resolved by adding `User www` to the top of the `httpd.conf` file. Also make sure the ownership of the biomart directory is `www`.

If Apache runs under a different user than `www` on your system, you should use that instead in the fix above.

2.4.5.3 No such file or directory (in the log)

The `martview`, `martresults` and `martservice` scripts can't find Perl.

Modify the first line of each script in the bin directory to point to the correct location of your Perl installation (which is often `/usr/bin/perl`).

2.4.5.4 Can't call method "settingsParams" on an undefined value (in browser)

Your `mod_perl` installation is broken or incompatible. Reinstall `mod_perl` from source and reconfigure MartView using the `--clean` option.

Note that on Mac OS X, installing `mod_perl` with `fink` may not be sufficient.

2.4.5.5 Exception::Class::Base::new (in browser)

Your `mod_perl` installation is broken or incompatible. Reinstall `mod_perl` from source and reconfigure MartView using the `--clean` option.

Note that on Mac OS X, installing `mod_perl` with `fink` may not be sufficient.

2.4.6 MartView maintenance tasks

2.4.6.1 Updating the existing Registry

If the datasets that your Registry points to have been updated or upgraded, but the list of datasets itself has not changed, then follow the instructions in this section.

If you modify your Registry in order to add or remove datasets, or need to change to a Registry file with a different name, you should refer to the section elsewhere in this document on switching to a different Registry.

Note if you rerun **configure.pl** without either the **update** or **clean** option it will use the default (**cached**) option to configure using the cached copy of the existing registry if it exists. This is only useful if you want to modify the server settings.

If the datasets that your Registry file points to have been updated to newer versions, you will need to reconfigure MartView. To do this you must first stop MartView, then change to the *biomart-perl* directory and type the following (replacing **myRegistry.xml** with your actual Registry file, just as you did when first configuring):

```
perl bin/configure.pl --update -r conf/myRegistry.xml
```

Answer **n** to the first prompt about configuring for the API, and answer **y** to the second prompt about keeping your existing server configuration. The dataset configurations that were downloaded previously will be checked and any that have changed will be downloaded anew. Finally, the various templates that define the MartView pages will be rebuilt and recompiled to match any changes found.

You can now safely start MartView up again.

2.4.6.2 Switching to a different Registry

If you alter the Registry file so that it points to different datasets, or decide to use a completely different Registry file, then MartView needs to be reconfigured from scratch using the new Registry.

If you have renamed the registry file then you can just rerun `configure.pl` using the new name:

```
perl bin/configure.pl -r conf/myNewRegistry.xml
```

If the altered registry still has the same name you must use the `clean` option to overwrite the cached copy:

```
perl bin/configure.pl --clean -r conf/myRegistry.xml
```

2.4.6.3 Changing memory usage

The default behaviour suitable for a server with a large amount of memory is to keep all the configuration data in memory (**--memory** option). If memory is an issue then you can run `configure.pl` with the **--lazyload** option which will store all configuration data locally on disk and just load what is currently required into memory. These options can be used with any of the above `configure.pl` options. For example to reconfigure the server in lazyload mode if none of the underlying configuration has changed you would run:

```
perl bin/configure.pl --lazyload -r conf/myRegistry.xml
```

2.4.6.4 Clearing download files

An optional setting in the *settings.conf* file in the *conf* folder allows users of MartView to request that their queries be run in the background and the results saved to file for them to download later. The location of the files is also defined in *settings.conf*.

The MartView system administrator needs to decide on a policy as to how long these files are kept on disk before being cleared out. Files can be removed safely just by deleting them.

2.4.6.5 Clearing log files

Log files are kept in the *log* directory. The logs are written by Apache, and can be maintained in the same way as Apache logs are maintained. In other words, you can pretty much do what you like to them.

Make sure you do not accidentally delete the *httpd.pid* file whilst clearing logs. If you do so, it becomes harder to stop MartView safely.

3 QuickStart Guide

3.1 Creating a mart

The first step is always to have some data to work with. You can choose either to format this data into tables yourself, or use MartBuilder to create a mart based on the contents of your existing relational database.

Whilst learning how BioMart works it is recommended that you work only with small datasets, enabling you to experiment easily and quickly with different settings.

3.1.1 Handmade marts

The simplest mart consists of a single table. It must follow a particular naming convention:

dataset_content_main

dataset is the name of your dataset. **content** is a description of what this table holds. The **main** suffix indicates that this is the main table of the dataset. Note how the three sections are separated by two pairs of underscores.

The table must contain at least two columns, one of which must be usable as a unique identifying key. This key column can be of any datatype but it must have a name that ends with the **_key** suffix. You can have as many other columns as you like, as long as you have a key column and at least one other. There can be no more than one key column on any given table.

To learn about the various BioMart naming conventions, see the section of this document about building marts.

An example SQL statement for creating a compliant table in Oracle would look like this:

```
create table mydataset_mycontent_main (  
    mypk_key int not null,  
    mynumber int,  
    mystring varchar2(20)  
);
```

Finally, you must ensure that this table has at least one row in it. None of the rows may contain a null value in the key column, but null values in other columns are allowable.

Your dataset is now complete and your schema contains a mart which is ready for configuring with MartEditor.

3.1.2 Auto-generated marts

If you have some data in a relational database which you wish to convert into a dataset, you can use MartBuilder to generate a script to perform the conversion for you.

MartBuilder is part of the *martj* package, which should be installed before you can run it. See the section on downloading and installing *martj* elsewhere in this document.

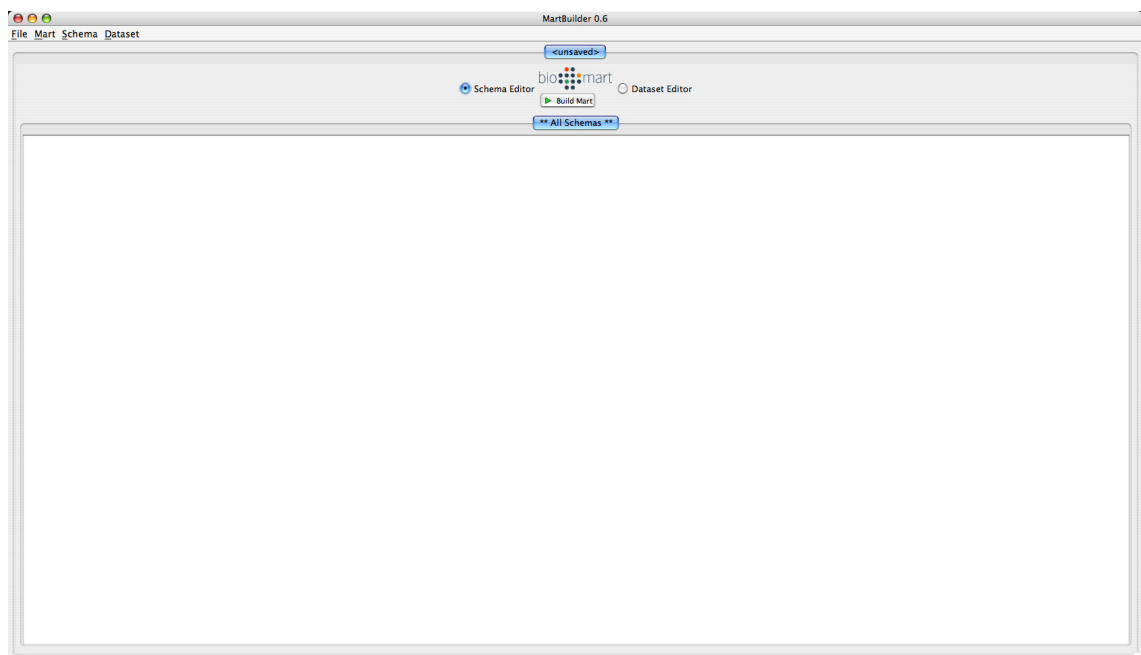
Start MartBuilder on Unix/Linux by changing to the *martj* directory, and typing:

```
bin/martbuilder.sh
```

On Windows you can navigate to the *martj/bin* directory and double-click on *martbuilder.bat*. On Mac OS X you can double-click on *MartBuilder* directly if you downloaded the binary distribution and did not compile it yourself.

A new MartBuilder window will open with an empty mart configuration already set up for you.

What you see here is a blank mart, waiting to have one or more source schemas added to it from which datasets will be generated later.



The next step is to connect to your existing relational database and add the schema which contains the data which you wish to transform.

3.1.2.1 Connecting to your database

Choose **Add Schema...** from the **Schema** menu. A new dialog box will open (the example given below has already had the fields filled in to connect to the Ensembl Variation database):

In this dialog box, enter a name for your new schema in the drop-down menu labelled **Name** at the top-left. Then, select a database type (MySQL, Oracle, Postgres etc.) from the **Database Type (Class)** drop-down. Enter the hostname of your database server in the **Host** field, and enter the database name and schema name in the **Database** and **Schema** boxes.

If you are using MySQL, then database and schema should be the same value, and should be set to the name of the database that contains the data you want to transform.

If you are using Oracle, then database should contain the SID, and schema should contain the name of the owner of the tables you will be transforming.

If you are using Postgres, then database should contain the database name, and schema should contain the schema name.

Finally, enter the database username and password in the **Username** and **Password** boxes. Click **Test** to see if you got your details right. If you didn't, go back and change them. If you did, then click **Add** to connect to the database.

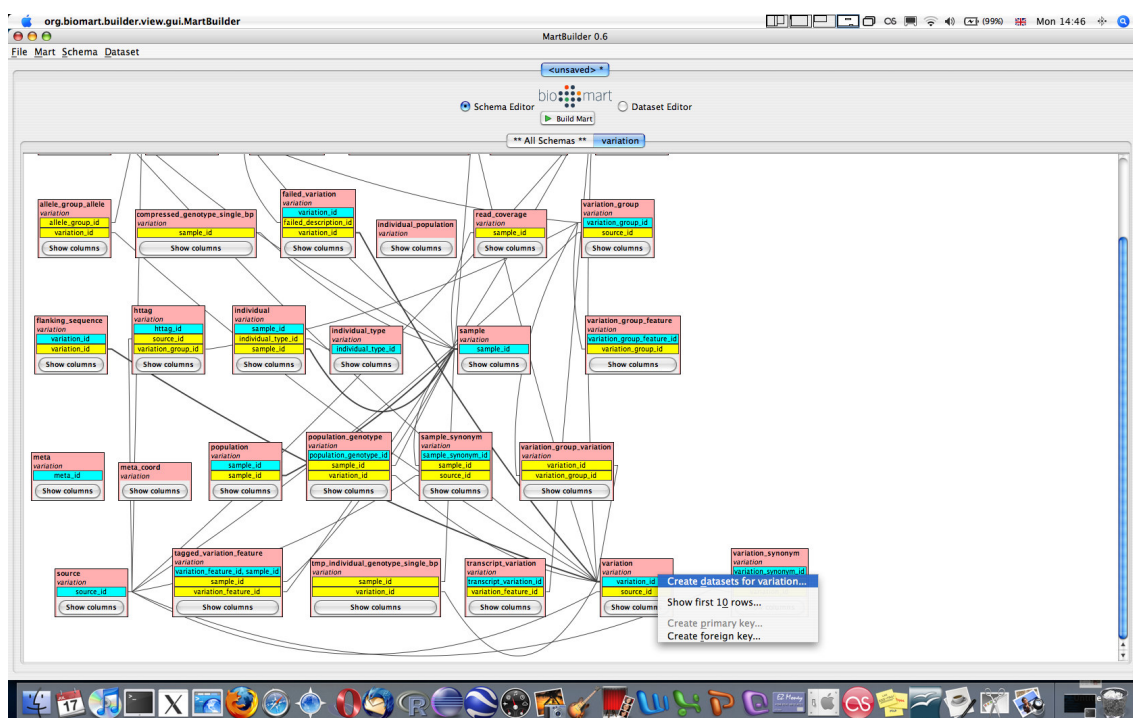
The MartBuilder window will gain an extra tab beside the **All schemas** tab, and will focus on that tab. The tab will have the same name as the schema you just added. Inside the tab will be a diagram describing your newly connected schema. You can scroll around this diagram to explore how the various tables are connected to each other. Relations shown with thin lines are 1:M whilst those with thicker lines are 1:1.

If your database does not enforce referential integrity, for instance if you are using MyISAM tables in MySQL, then MartBuilder attempts to guess how they are related. You are free to modify the relations it displays if you think it has got it wrong.

3.1.2.2 Designing a simple dataset

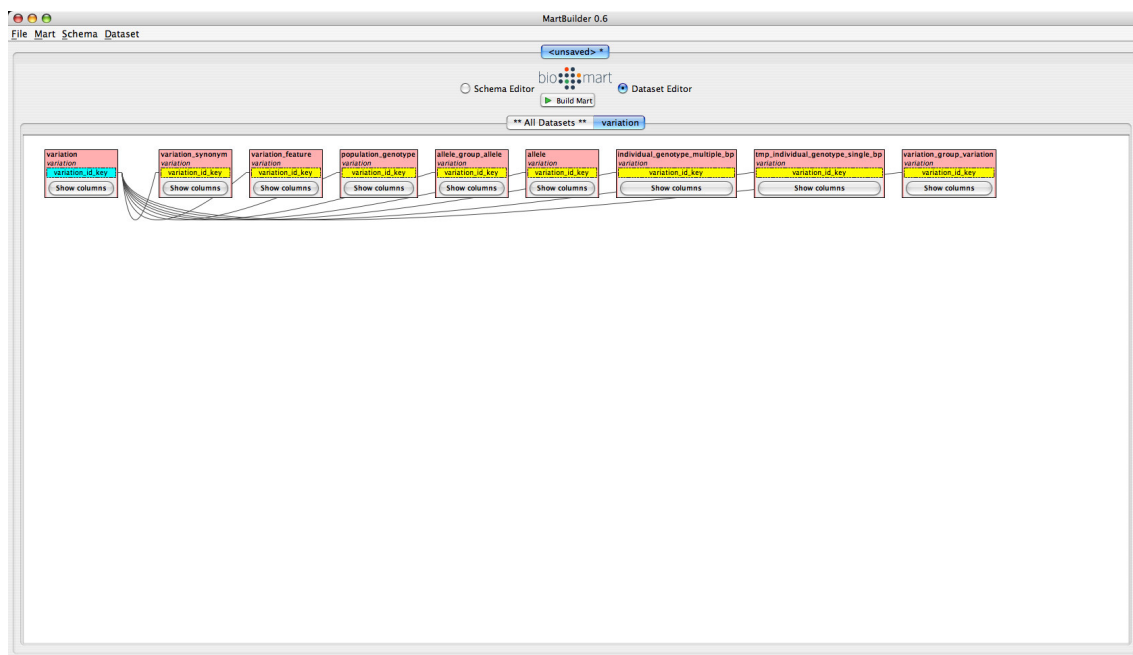
Find the table that contains the data you wish to use in the main table of your dataset. You can either scroll to it, or right-click anywhere on the background of the diagram and choose **Find table...**

Right-click on the table once you have found it, and choose **Create datasets...** (the menu item will include the name of the table you have clicked on):



In the box that pops up, just click **Create** to continue for now.

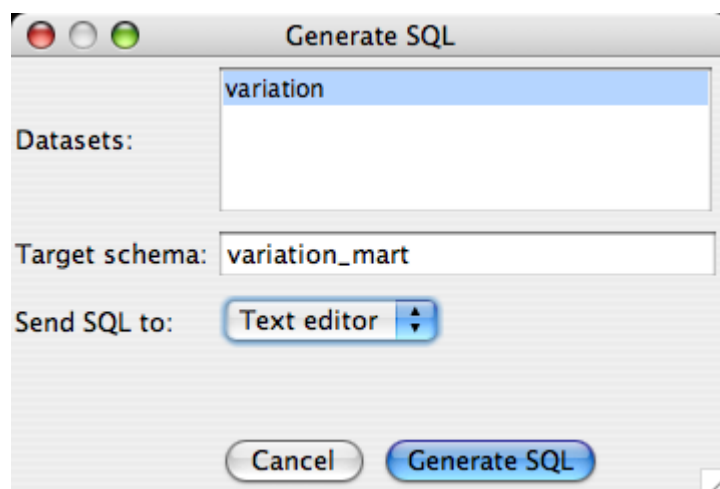
The window will change to select the **Dataset Editor**, and you will see a tab labelled **All Datasets**, and another tab containing the name of your new dataset. The tab named after your dataset will be selected and a diagram of the transformed dataset will appear inside it. On the left is the main table, and on the right will be one or more dimension tables:



Your dataset does not actually exist yet. In order to create it, you need to generate the SQL for it then execute that SQL against your database.

3.1.2.3 Saving the SQL

Choose **Generate SQL...** from the **Mart** menu. The **Generate SQL** dialog box will appear:



In the **Target Schema** field, enter the name of the database schema in which you wish to create the dataset. This will normally be different to the database schema that contains the tables you are intending to transform, but must exist in the same database server instance.

Make sure the **Text editor** option is selected in the drop-down menu in this dialog. Then, hit the **Generate SQL** button.

Depending on the size of your database, the generation may take some time. A message will pop up when it is completed. If something goes wrong, a different message will pop up explaining what the problem is.

After a while, your SQL will appear in a new window.

If you are using Oracle, you will need to grant permissions so that the source schema owner can issue create/alter/insert/update/drop table/index statements against the target schema that will contain the transformed tables. You must do this before attempting to execute the generated SQL.

3.1.2.4 Executing the SQL

From the window containing your SQL, use the buttons provided to save it as a text file. Then, locate that text file and use your usual database connection tools (*sqlplus*, *mysql*, *psql* etc.) to connect to the schema that you wish to create the dataset in and execute the SQL in the file. The script will issue a series of statements that create the new tables for your dataset by selecting and transforming data from the schema containing your original tables.

Alternatively you can execute the SQL directly from MartBuilder by sending the SQL to the MartRunner tool, either from the menu in the **Generate SQL** dialog box or by clicking the **Build Mart** button in the top panel of MartBuilder. Progress can be monitored using the **Monitor MartRunner progress...** option in the **Mart** menu. Note that MartRunner must already have been started (`bin/martrunner.sh 1234`).

Your schema now contains a mart with a complete dataset ready for configuring with MartEditor.

3.2 Configuration using MartEditor

Once a mart has been built using MartBuilder or by other means, it will need configuring using MartEditor before it can be used by MartView or the Perl or Java APIs.

Like MartBuilder, MartEditor is part of the *martj* package, which you will need to have installed before you can use it.

To start MartEditor, change into the *martj* directory and type (on Unix/Linux):

```
bin/marteditor.sh
```

On Windows or MacOSX, navigate to the *martj/bin* directory and double-click on *marteditor.bat* or *MartEditor* as appropriate (*MartEditor* option only available if you downloaded the binary distribution and did not build *martj* from source).

You may get an exception on starting MartEditor which complains that it cannot connect to your database.

If you have previously connected, it will probably be because you did not save your password in the connection settings. Use the **Database Connection** menu option to enter it again.

A blank MartEditor window will open. Use the **Database Connection** option in the **File** menu to establish a connection to the database in which your mart lives. The window that opens will look like the one below. In the example shown, an Oracle connection is being established:

The fields in this dialog are equivalent to those in the MartBuilder **Add Schema** dialog, and you should enter similar values, except this time they should reflect the schema that your mart lives in, as opposed to the schema that your original data lives in.

After clicking **OK**, MartEditor will connect to your mart.

Use the **Naïve** option in the **File** menu to establish a basic configuration for the dataset in your mart. MartEditor will ask you to choose from a list of available datasets which one you wish to configure. There will probably only be one.

After selecting a dataset and clicking **OK**, a window should open containing details of the configuration for that dataset. To save the configuration, choose the **Export** option from the **File** menu.

Your mart and associated dataset are now configured and ready to run queries.

3.3 Setting up MartView

You will need to download *biomart-perl* first. Do not configure it yet.

Create a Registry file in the *biomart-perl/conf* directory, called something like **myRegistry.xml**. The file should contain the following, substituting the various database settings for the settings that describe the schema where your newly created mart lives:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MartRegistry>
<MartRegistry>
  <virtualSchema name="default">
    <MartDBLocation
      name           = "example"
      displayName    = "My Example"
      databaseType   = "mysql"
      host           = "localhost"
      port           = "3306"
      database       = "mymart"
      schema         = "mymart"
      user           = "myusername"
      password       = "mypassword"
      visible        = "1"
      default        = ""
      includeDatasets = ""
    </MartDBLocation>
  </virtualSchema>
</MartRegistry>
```

The **name** and **displayName** options dictate how the mart will appear to end-users. The **name** is used for referencing this mart from applications such as MartShell. The **displayName** is used when displaying this mart in applications such as MartView. You do not need to worry about the **visible**, **default**, and **includeDatasets** options for now. You can safely leave them as they are in the example above.

One Registry file can include marts from many different locations and of many different types. However, MartView can only be configured to use a single Registry file at a time.

The next step is to configure MartView.

MartView can only be configured if you have a usable Apache and ModPerl installation. If you do not, you will need to follow the full instructions on installing MartView to find out how to set these up.

Configure MartView by typing:

```
perl bin/configure.pl -r conf/myRegistry.xml
```

Accept all the default answers for the questions it asks you, except maybe for the hostname and port number on which MartView will listen if you don't like the default (localhost:5555).

To start MartView, wait for the configuration step above to complete then from the *biomart-perl* folder type:

```
/usr/sbin/httpd -d `pwd`
```

(Substituting */usr/sbin/httpd* for the actual path to your Apache binary).

If you have previously attempted to configure MartView and this is your second or subsequent attempt, follow the instructions elsewhere for switching to a different Registry instead.

MartView can now be used to run queries against your dataset, via both the web browser interface and the web services API. You will find it listening for requests at the hostname and port you configured above. By default this will be:

```
http://localhost:5555/biomart/martview
```

4 Building Marts

4.1 Structure of a Mart

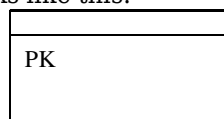
A mart is a collection of datasets. It is nearly always synonymous with a database in MySQL, or a schema in Oracle and Postgres.

A dataset is a collection of tables that follow a given naming convention. The table naming convention is **dataset_content_type**, where **dataset** is the name of the dataset, **content** is a free-text summary of the contents of the table, and **type** is either **main** (for main tables) or **dm** (for dimension tables), both of which are explained below. Double underscores (__) are meaningful and must not be used elsewhere within the table name except as shown above.

BioMart software does not recognise case-sensitive names, therefore your table and column names must be such that they work correctly in SQL statements without needing to be quoted or escaped.

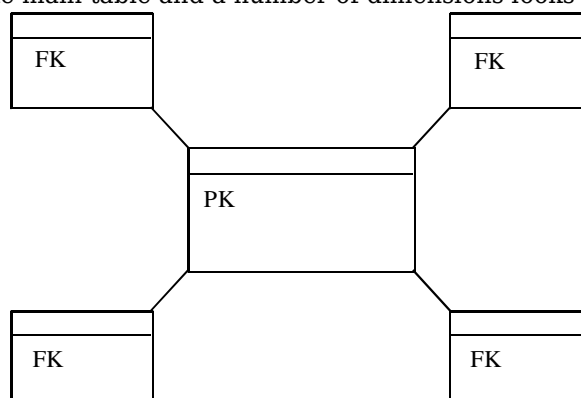
Each dataset must have at least one single central table called the main table, with a type of **main**. This main table is involved in all queries, and will normally contain the information most frequently requested. It must have one column ending in the suffix **_key** which contains a unique identifier for each row, similar in function to a primary key.

The simplest dataset therefore looks like this:



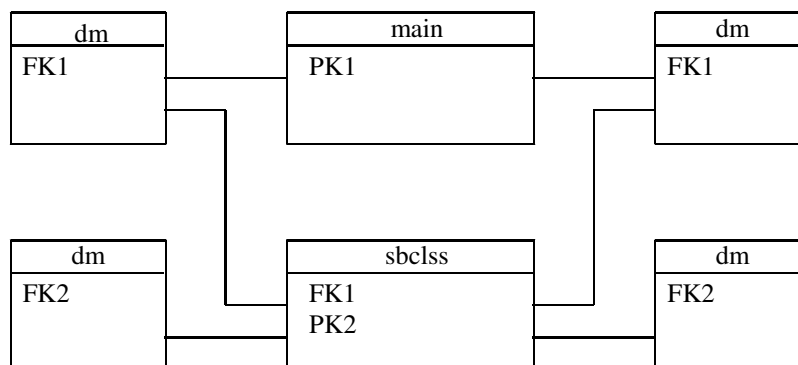
A dataset may optionally have a number of dimension tables containing satellite information related to the main table. These dimension tables are recognised by having a type of **dm**. Each dimension table must have a column that contains values from the **_key** column of the main table to which the data in the dimension table is related, similar in function to a foreign key. The name of the column must be exactly the same as the one on the main table that it refers to. Dimension tables must not have any other columns ending in **_key**.

A dataset with a single main table and a number of dimensions looks something like this:



Some datasets may choose to include more than one main table in order to speed up queries. In such cases the first main table contains a subset of columns from the second main table, which is known as a subclass table. The second main table can itself be subclassed by a third main table, and so on, creating a subclassing chain. Subclassed main tables must contain their own unique **_key** column as well as all **_key** columns from the parent main table which they are extending. All subclassed tables use the **main** type in the naming convention.

A subclassed dataset looks a bit like this diagram, showing how all dimensions of the main table can also be referenced from the subclassed table:



The set of all columns from all tables in a dataset is equivalent to the set of attributes available on that dataset. Every filter in a dataset is created by restricting an attribute to a particular value or range of values. Therefore filters are like the where-clause in SQL statements and attributes are like the columns listed in the select portion of a SQL statement.

When configuring datasets with MartEditor, note that it does not depend on formally declared primary and foreign keys in datasets but on the column naming convention instead. The links between various tables in datasets are derived from matching column names containing a **_key** suffix.

This logic is separate from that used by the MartBuilder key-guessing algorithm as it applies not to source schemas but to the end-product datasets. See the MartBuilder section of this document for details on how key-guessing works.

4.2 Data model

BioMart software will work with any set of tables as long as they follow the required naming convention. However, it is recommended that you follow the following guidelines when designing your dataset:

1. The dataset main table contains the data about the central dataset object which is to be retrieved (e.g. Gene). This includes all the data related to this object as 1:1 or n:1.
2. The dataset dimension tables store all the data referenced by the primary key of the main table and related to it as either 1:m or m:n
3. The subclass tables are always 1:n to the main table.

4.3 MartBuilder

The easy way to transform the data contained within a relational database is to use MartBuilder. MartBuilder allows you to select the central table of interest in a database and automatically design datasets based around that central table. It follows the transformation guidelines and naming conventions outlined above and will generate SQL statements that you can execute to perform the actual transformation.

4.3.1 Starting

On a Mac, if you downloaded the binary distribution then MartBuilder can be started by double-clicking the *MartBuilder* application icon.

On Windows, you can double-click the *martbuilder.bat* file in the *bin* folder of the *martj* distribution.

On Unix/Linux, you need to change to the folder containing your *martj* distribution and type:

```
bin/martbuilder.sh
```

In all cases this should result in the display of an empty MartBuilder application window, with a new, blank mart opened.

4.3.2 Schema editor vs. Dataset editor

A source database is known as a schema.

The **Schema Editor** button shows you all the schemas which MartBuilder is currently using to provide tables for the datasets you are designing. Changes to any schema whilst this button is selected will affect all datasets that use tables from the altered schema. The **All Schemas** tab shows you all the schemas currently known, and each individual schema also has its own tab beside the **All Schemas** tab.

The **Dataset Editor** button shows you the datasets you are designing. The **All Datasets** tab shows you all the datasets currently being designed, and each individual dataset also has its own tab beside the **All Datasets** tab.

4.3.3 Connecting to a source database

Using the **New** option from the **File** menu, start a new mart. Alternatively, use the default blank mart that opens when you start the application. Make sure the **Schema Editor** button is selected.

You can have several marts loaded at once if you like, with each represented as a tab across the top of the MartBuilder application window.

Choose **Add...** from the **Schema** menu. A new dialog box will open allowing you to specify connection settings for your schema. The **Name** box at the top-left is for you to enter a unique name by which MartBuilder will refer to this schema. If you have used MartBuilder before, you will find previous settings you have entered are available by browsing the drop-down menu under the **Name** box.

MartBuilder supports MySQL, Oracle and Postgres. It ships with JDBC drivers for all three, but you may choose to use your own driver. In this case you should leave the **Database Type (Class)** drop-down box blank and use the area beside it to identify your alternative JDBC driver. That driver must be on the classpath before you start MartBuilder else it will not be found. You will also have to manually specify the **JDBC URL** connection string.

You can also specify JDBC drivers for other database platforms but MartBuilder will refuse to generate SQL for them. Please contact the BioMart team if you need to use another database platform, and we will work with you to write the necessary extension module to add support for that platform to MartBuilder.

Click the **Add** button to add your schema to the mart. It will take a short while to read the database metadata and work out what tables and columns are available. Eventually, a new tab containing the same name will appear to the right of the **All Schemas** tab and become focused.

4.3.4 Multiple source schemas (Partitioned Schema)

If you have multiple source schemas containing very similar tables and you want to produce one mart design that can be applied to all of them, then this can be achieved by using the partitioned schema feature.

This feature requires knowledge of Regular Expressions (regexes).

First, you should enter the details of just one of the schemas in the normal way as described above. Before clicking **Add**, perform a couple of extra steps to introduce the remaining schemas.

In the schema connection box, there are two fields near the bottom **Schema matching regex** and **Schema naming expression**. The regex box should contain a regular expression which will match all the schemas you want to include. For instance, if you want to include all schemas that start with the word `ensembl`, you should use this regex:

```
^ensembl (.*) $
```

This regex matches all schemas starting with the word `ensembl` and assigns the remainder of the schema name to the first bracketed group.

Then, in the expression box, you need to type an expression that will generate a unique name for each schema. This refers to the bracketed groups selected by the first regex by using `$1` for the first group, `$2` for the second, etc. The values generated by this expression will be used to prefix the names of the generated datasets.

In our example, if we used the expression:

```
$1
```

and our source schemas looked like this:

```
ensemblHuman
ensemblMouse
ensemblRat
```

then our generated datasets would have the following naming patterns:

```
Human_<dataset>__<table>__<type>
Mouse_<dataset>__<table>__<type>
Rat_<dataset>__<table>__<type>
```

These generated names from the expression are the schema partition names. You can see what they will look like by examining the table at the bottom of the schema connection dialog. You can change them at any time by using the **Update...** menu option from the **Schema** menu.

There is a dropdown menu beside the **Schema Editor** and **Dataset Editor** radio buttons. By default this menu shows *****All Partitions*****, which means that if you are using partitioned schemas, it shows you all the tables from all the schemas combined. It's a bit like drawing them all out on transparent plastic sheets then laying them in a pile with the matching parts lined up on top of each other.

If you select this dropdown menu you will see it contains a list of all the generated names from the expression field in the connection dialog. Selecting any one of those names will modify the diagrams to show only the parts of the schema that exist in the selected schema partition. It will also remove all parts of datasets that rely on tables or relations that do not exist in the selected schema partition.

4.3.5 Adjusting the schema

Clicking on the tab representing your schema will show you a diagram detailing the tables, columns and relations available in that schema.

The metadata reader will have guessed the relations and foreign keys if they were not defined in the metadata, which is usually the case for users of the MyISAM table type in MySQL. This process is called key-guessing. Key-guessing is turned on by default wherever the metadata returns no relations. You can turn key-guessing off using the context (right-click) menu on the white background area of the schema diagram.

Key-guessing defines foreign keys as columns whose name is the same as the primary key of some other table, optionally suffixed with **_key**. When doing this it ignores any primary keys that do not match the name of their table, optionally suffixed with **_id**, or that are simply called **id**. It then creates relations between these guessed foreign key columns and the primary key of the matching table.

This approach reduces the number of false-positive matches. A small number of relations correctly suggested are easier to manage and modify than a larger number of incorrectly suggested ones.

If you see no relations even though key-guessing is on, then you will have to define them manually.

By default, all relations that link a primary key (PK) with a foreign key (FK) where a PK containing identical columns to those of the FK exists on the FK's table will be defined as 1:1. All other relations are 1:M. MartBuilder will allow the user to define relations as M:M in certain circumstances but will treat them as 1:M during transformation, with the 1 end being the end of the relation first reached by the transformation algorithm.

1:M and M:M relations are displayed using thin lines. 1:1 relations are displayed using thicker lines.

The user can change the cardinality of a relation by using the context menu on the relation. The user can also mask unwanted relations and any relations that the metadata reader has guessed incorrectly, so that the transformation algorithm does not include them.

Normal relations show up in black. Relations that are masked show up in red. Relations that the user has established themselves either using the context menu on the relations or by dragging keys onto other keys will show up in light green.

All changes to the schema will result in dark green outlines appearing around the affected tables and relations in the schema, and around any new tables, columns or relations created as a result of the changes in datasets that use that schema.

Changes are defined as anything you do, or anything new introduced by using the **Update schema** menu options to synchronise the schema diagrams with the contents of the database.

Use the **Schema** menu to **Accept** or **Reject** the changes and make the dark green outlines go away. Accepting the changes does nothing except remove the colouring. Rejecting the changes will mask any new columns or tables introduced into any datasets as an effect of the change.

Keys can also be defined by the user, or redefined if the metadata reader or key-guessing algorithm got it wrong. This is all done through the context menu on the keys.

Normal keys are outlined in black. Keys that are masked (because they are incorrect or unwanted) show up outlined in red. Keys that the user has created or modified using the context menu will show up outlined in light green.

All masked tables and relations can be hidden from the schema diagram by using the **Hide masked** checkbox in the top-right corner of the diagram.

The columns on tables are hidden by default. The list of columns on each table can be toggled on and off by using the **Show/Hide Columns** button.

If a database schema is changed and you wish to bring the MartBuilder copy up-to-date, use the **Update Schema** item on the **Schema** menu to check the connection settings then synchronise it. Synchronisation will remove any relations, keys, columns and tables that no longer exist in the database (except for any that were created or modified by the user). It will then add in new objects from the database that were not already shown in MartBuilder's schema diagram. The **Update all schemas** menu option in the **Schema** menu will do this for all currently open schemas using the existing connection settings for each schema.

4.3.6 Creating a dataset

First you need to identify your central table of interest. Locate it in the schema diagram, bring up the context menu for it and choose the **Create datasets** option. You can right-click anywhere on the background and select **Find table** if you are having problems finding a particular table in a complex schema. If you have a number of candidate central tables and are not sure of the best way of selecting one from amongst them then the dialog box that appears when you do this will allow you to select multiple tables from a list.

MartBuilder will use the selected tables in the dialog box to work out possible combinations of main and subclassed main tables which include all the selected tables somewhere in the dataset. Whether they will be main tables or subclassed tables is up to the algorithm. If it is not possible to create a single dataset with all the selected tables as main or subclass tables then a number of possible datasets will be created, each one choosing a different table from the list for use as the main table.

The **Dataset Editor** button will be automatically selected after dataset creation, as will the tab within it representing your new dataset. The tab will show the name of the dataset. The name of the dataset is equivalent to **dataset** part of the table naming convention. It will most likely be the same as the name of the table you clicked on in order to generate this dataset, assuming this is a valid dataset name and does not clash with another existing dataset design.

Note the columns in your dataset all have numeric suffixes. These are a reference to the exact schema and table from which each column came. They help you track where columns move to when you alter the source schema. You can of course rename them to anything you like, including just removing the numeric suffix.

4.3.7 Adjusting the dataset

You can explore the dataset design by clicking on the tab in the Dataset Editor with a corresponding name. It will show you the main table outlined in black, subclassed tables outlined in red, and dimensions outlined in blue. The main table is at the top-left of the diagram with dimensions of that main table arranged to its right. Subclass tables appear below the main table down the left side of the diagram, each with their own dimensions to their right.

Any changes to the schemas backing the dataset will result in dark green outlines appearing around any new tables, columns or relations created in the dataset as a result of the changes.

Use the **Dataset** menu or the context menu over individual tables in the dataset diagram to **Accept** or **Reject** the changes and make the dark green outlines go away. Accepting the changes does nothing except remove the colouring. Rejecting the changes will mask the new columns and/or dimension tables.

Individual dimensions can be removed from the dataset by masking them. They will appear to be grey in colour. Masked dimensions will not have SQL generated for them. Relations between masked dimensions and their parent main or subclass table will appear in red.

All masked dimensions and relations can be hidden from the dataset diagram by using the **Hide masked** checkbox in the top-right corner of the diagram.

The **Explain Dataset** menu option in the **Dataset** menu allows you to explain the dataset. This will pop up a dialog which is similar to the content of the **Schema Editor**, except that the changes you make here are specific to this dataset and will not affect other datasets. Using the context menus in this dialog you can place restrictions on tables and/or relations, mask out certain tables or relations so that they are not used in the dataset, or specify relations that represent subclassing. You can also specify that a relation can be followed multiple times (compounded).

The context menu on each individual table in the dataset will allow you to **Explain table**. This will pop up a two-part dialog. One part of this dialog shows the schema diagram again, but with only the schema tables involved in providing information for this dataset table highlighted. The other part shows you in which order they are joined, and which columns in the dataset table result from each join made. The steps which show grey (masked) tables are there for illustration of potential extension points to the transformation and do not actually get involved in the transformation unless you perform actions which bring them in.

You can hide these masked tables using the **Hide masked** checkbox at the top of the diagram.

The context menu on the tables and columns allow you to rename them. Columns that are involved in any key will insist that their name ends in **_key**. Columns in keys at one end of a relation will always update to have the same name as the column at the other end. The name shown for the table is equivalent to the **content** part of the table naming convention. It also allows you to add extra columns which contain expressions evaluated from the values in other columns on that table. These expression columns accept SQL expressions that are most likely going to be specific to the RDBMS platform your data is on.

If you want to apply your dataset to multiple source schemas, producing identical transformations but with different dataset names, you need to investigate the **Partitioned schema** menu option in the **Schema** menu. This will affect all datasets that use that schema. See the section above on adding schemas for details on how to do this.

The various menu options available for datasets and dataset tables and columns are described more fully later in this document.

All changes you make are instantly reflected in the dataset diagram. There is no undo option.

4.3.7.1 Optimiser columns

Boolean and count columns (known as optimiser columns) are extra attributes on a dataset that allow the user to discover how many rows in a dimension table match a row in a main or subclass table. Boolean columns contain 1 or 0 (or 1 or null, depending on your choice), for presence or absence, whereas count columns contain a number indicating the number of matching rows, which could be 0. These columns can either be attached to the main table itself, or collected together in a new dimension related to the main table. This is all done through the **Optimiser column** submenu in the **Dataset** menu. Main and subclass tables will always have a count column for their immediate child subclass table, if any.

If you turn optimiser columns on, then they will apply to all dimensions and subclasses. To turn them off again for individual dimensions or subclasses, use the **Skip optimiser** context menu on that table. By default all optimiser columns are indexed. If you wish to skip indexing a particular dimension or subclass, right-click on it and use **Skip index optimiser**.

If you want to produce separate optimiser columns for a dimension based on what values appear in a particular column of that dimension, use the **Split optimiser...** context menu over the column from which you wish to get the values. The dialog will allow you to choose an optional additional column to concatenate values from for use in the optimiser column, with an optional separator to go between each value. There is an option allowing you to specify the data size of the optimiser column. By default the optimiser columns will include a prefix indicating the dimension name and a suffix indicating the optimiser type (count or bool). You can turn these on and off using the checkboxes in the dialog. You can also choose to

The split optimiser is a hard concept to understand but one example of its use would be to have multiple optimiser columns for various strains of a species, and place into the optimiser column the symbols of the alleles found in a particular variation for that species (e.g. 'A/T/G').

4.3.7.2 Partition Tables

When you create datasets from the schema diagram, you'll notice there are actually three different Create menu options. The first, **Create dataset**, is the one you use to create datasets. The second, **Create partition table**, is used to create partition tables (this section). The third, **Create unrolled dataset**, is used to create unrolled datasets, which are a kind of recursive thing useful for ontology representations (next section).

A partition table is a special kind of dataset that shows up in red in the **All datasets** tab. It is used to generate a virtual dataset, which never gets SQL built for it. The virtual dataset instead is used to generate lists of values which can then be used to divide up, or partition, other datasets or dimension tables within those datasets.

A partition table is built and modified in exactly the same way as a dataset. Any existing dataset can be converted into one by using the **Convert to partition table** menu option in the **Dataset** menu. By modifying a partition table, you modify the columns that are available to select values from later.

The partition table dialog, accessed via the **Convert to partition table** menu option in any partition table, shows a list of columns at the top-left, and allows you to move columns into the second list to the right. The columns in the second list are the ones that will produce the values for the partition table entries. Any one of these columns can have a regex applied to modify the data within it by selecting the column in the righthand list and using the text boxes to the right to enter a regex and expression to convert the data. This is useful, for instance, for abbreviating longer values for use elsewhere in table names.

Under the column selection boxes you will see a table showing the first few rows of data produced by your selected columns. This is useful for previewing the data your selections and regexes will produce.

There is a special column, called `__SUBDIVISION_BOUNDARY__` which can be added only once to any list of columns. This essentially divides one partition table into two related partition tables, with a set of parent values on the left of the boundary, each of which has one or more associated child values on the right of the boundary.

When partitioning just a dimension, you do not need to use `__SUBDIVISION_BOUNDARY__`. When partitioning just a dataset, you also do not need to use it. When partitioning a dataset and the main table of that dataset simultaneously from the same partition table, then you need to use it, as the lefthand (parent) side will apply to the dataset and the righthand (child) side to the main table.

The partition table can then be applied to a dataset or dimension, or a dataset and main table combination.

To apply it to a dataset, use the buttons in the dialog to select one, or choose **Partitioned dataset...** from the **Dataset** menu. The dropdowns at the bottom right will ask you which column in the partition table is supplying the values to divide the dataset up with, which column in the dataset should be used to search for rows that match the values, and which column from the partition table is providing the names to be used to prefix each partition of the dataset. The result will be one dataset for each unique set of values in the partition table, with each dataset prefixed with a unique name based on the selected column.

To apply to a dimension, again use the buttons, or choose **Partitioned...** from the context menu of a dimension. Follow similar steps, but this time the unique names for each dimension will be inserted between the table name of the dimension and the `__dm` suffix.

If you wish to apply the partitioning to both dataset and main table simultaneously, apply to the dataset initially and after adding in `__SUBDIVISION_BOUNDARY__` to the selected columns you will see a second set of dropdowns in the dialog to select the details for the main table.

If partitioning a dataset and dimension in that dataset simultaneously from the same partition table, it is a good idea to use the **Update partition counts** menu option in the Dataset menu before generating SQL to ensure that the correct number of tables are produced.

4.3.7.3 Unrolled Datasets

An unrolled dataset is one where a recursive relation exists between two tables in the source schema, and you want this to be expanded into a number of distinct rows of data in the dataset. The rows define relationships between every node and every possible descendant (or ancestor) of that node. This is useful for things like ontologies where you have the need to quickly identify if a term is an ancestor or descendant of another term no matter how distantly related the two may be.

To create one, use the Create unrolled dataset menu option over the table containing your nodes (the terms of an ontology, for instance). The dialog that opens will contain that table in the first dropdown. If you chose the wrong one, change it in the dialog. The next two dropdowns in the dialog ask you to select the unique identifier of the node (usually the primary key) and the column that names or describes each node. The fourth dropdown asks you to choose the table which defines the relations between pairs of nodes. The fifth and six dropdowns ask for columns on that table which refer to the unique identifiers of the two nodes involved. The generated dataset will contain all mappings from the parent identifier to itself and also to its children and its children's descendants. If you swap the parent and child identifier columns over in this dialog, you will get a dataset that maps in the opposite direction, i.e. to all ancestors rather than descendants.

The final option, the checkbox asking you whether or not to **Reverse sense**, is a little complicated. The default algorithm assumes that the second table you selected maps child nodes to parent nodes, in which case leave the box ticked. However, if the table maps in the opposite direction, from parent to child, then untick the box. The difference between the two situations can be determined by examining the node table you selected. If the entries in it that have no entry in the parent column you selected in the relationship table are nodes with no children or parents, then tick the box. If they are nodes that have children but no parents, then untick the box. (This applies to descendants – swap the words child and parent in this paragraph if you want to get ancestors instead).

4.3.7.4 Loopback Wizard

One entry in the dataset table context menu is the Loopback Wizard. This sets up a combination of options to allow the dataset table transformation to include a transformation path twice, using the values from a column to differentiate between the two paths. This is useful if you have an $X1 \rightarrow Y1 \rightarrow Z12 \rightarrow Y2 \rightarrow X2$ type data structure in your database, where each Z12 has a Y1 and a Y2, each of which has their own X. The resulting table after using the wizard will contain one row for each Z, containing both Y1 and Y2 and both associated Xs.

The wizard works by asking you to choose a turning point' from the tables currently involved in the transformation on the selected table. This is table Z in the above example. It will then

ask for a column to use to differentiate between the two branches. This will be a column from Y in the above example that is guaranteed to not be the same in Y1 and Y2.

The loopback wizard will then apply all necessary compound relations and loopback relations in order to construct the table this way.

The loopback wizard will not work if you have used the Start transform from here to change the transformation start point (see later). If you have done this and still want to use the loopback wizard, you have to reset the transform start point to the default setting, use the loopback wizard, then reinstate the transform start point change.

4.3.7.5 Recursive Subclass

This is an option available on subclass tables. In effect it delegates to the compound relation option on the relation that leads between the underlying tables of the main and subclass tables in the source schema diagram.

This is a very special case that allows a recursive relation between the main and subclass tables to be unrolled to a specified depth, as given in the dialog used to set it up.

Imagine that your main table has features in it, and that they are nested – each feature has child features which are also defined in the same table. A second table defines the relations between each feature via a pair of 1:M relations from the main table.

You would first create a dataset based around the main table, then create a subclass table on the table that defines the relations between features. This will provide a simple parent-main child-subclass setup.

By using the recursive subclass option, you can increase the number of times the loop between parent and child features is followed. The resulting dataset will contain N subclass tables, chained to each other in a 1:M chain from the parent main table downwards, where N is the number you specified in the dialog. Each subclass table will contain the child features of the features described in the previous subclass or main table.

4.3.8 Generating SQL

Once you are happy with your dataset design, or set of dataset designs, you can use the **Generate SQL** option from the **Mart** menu to save the SQL for your mart. The dialog that pops up allows you to select which datasets you want to generate SQL for and whether you want to view that SQL on screen, save the SQL to disk or send the SQL to MartRunner (see section 4.4). The target schema name should be the name of the schema (or database in MySQL) which will contain the tables for the generated mart and datasets. SQL will be generated to be compatible with the RDBMS platform from which the schemas used in the dataset come.

If you are using Oracle, you will need to grant permissions so that the source schema owner can issue create/alter/insert/update/drop table/index statements against the target schema that will contain the transformed tables. You must do this before attempting to execute the generated SQL.

If you are using partitioned schemas, you will see a list of source schema partitions. By default the entire list is selected, meaning that SQL will be generated for datasets based on

all the selected schema partitions. If you want only one or a few, but not all, change the selection in the list to reduce the number of datasets that will be generated.

If the schemas come from different RDBMS platforms, then SQL generation is not possible.

If the schemas come from the same RDBMS platform but running on different instances or servers, then SQL generation will also not be possible unless you have established some kind of link between the instances or servers that make the various source schemas accessible by name from a statement run in any other schema.

The **Text editor** option will present a simple text editor window from which you can copy-and-paste statements to another application or save the SQL to a single file.

The **File** option will generate a zip file on disk. The zip file will contain a series of folders, and a `MANIFEST.txt` file. The `MANIFEST.txt` file describes the order in which to run the various files contained in the folders. They must be run in that order as there are many dependencies between statements which will fail if not run in the correct order.

The **MartRunner** option will ask for a hostname and port number to connect to MartRunner and send it the SQL for later execution. See the section on MartRunner elsewhere in this document for details on how MartRunner works, and how to monitor it.

If you know you want to send your generated SQL directly to MartRunner, you can avoid a few clicks by using the **Build Mart** button in the tab belonging to the mart you want to build SQL for. The button will open the dialog with the MartRunner option preselected. The button is located just below the BioMart logo.

4.3.9 Saving your work

Files saved using MartBuilder 0.6 will open in 0.7, but dataset definitions will not be fully preserved as the internal naming system for describing schemas has changed between the two releases.

The **Save**, **Save as** and **Open** options in the **File** menu allow you to save your current schema and dataset designs to file for later retrieval. The file will relate to the currently selected mart and will contain all schemas and datasets defined within it, including all connection settings.

This means that once you have refined your schema design and defined your dataset design and saved them to file, you do not need to do anything more than open the saved file and use the **Update schema** function whenever your source database schema changes, rather than having to recreate the design from scratch.

Another bonus is that MartBuilder does not require a connection to the database in order to open these files. Therefore the designs can be edited offline. A database connection is only required when establishing a new schema, synchronising an existing schema, or generating SQL statements.

The file format used is a custom XML format specific to MartBuilder. It is not advisable to edit these files by hand, but being XML it is possible to do so if you should so wish.

4.3.10 MartBuilder options in full

Each option is associated with a particular situation or context within the MartBuilder application. In the table below these contexts are abbreviated as follows:

S - Schema View

AS - the All Schemas tab in Schema View

D - Dataset View

AD - the All Datasets tab in Dataset View

ED - Explain Dataset view

ET(S) - Explain Table view, schema tab

ET(T) - Explain Table view, transformation tab

Options that appear in the Schema view affect all datasets which use that schema. Options which appear in the Explain Dataset view affect all tables within that dataset. Options which appear in either part of the Explain Table view affect only that table.

The Explain Table view is reached by right-clicking on any table in the Dataset View and choosing the appropriate menu option.

All option changes have instant effect and will cause all datasets affected to be adjusted accordingly. ***There is no undo feature.***

Option	Context	Description
Adding schemas	S	Add a schema by selecting the appropriate option from the Schema menu.
Removing or renaming schemas	S	Remove/rename a schema by selecting its tab then choosing the appropriate option from the Schema menu, or right-clicking on the tab and removing/renameing it from the context menu.
Renaming schemas	AS	Double-click on the schema name in the All Schemas tab. Edit the name. Hit <code>Enter</code> to accept it, <code>Escape</code> to cancel the change.
Replicating schemas	S	Choose the schema tab, then use the context menu in the tab or in the Schema menu to replicate it. Replication will create an entire copy of the schema including all modifications you have made to it, and will allow you to point that copy at a different database.

Partitioning schemas	S	<p>Use the option described below to update the schema (or this can also be done when adding the schema initially).</p> <p>The two extra text boxes in the schema dialog are regular expressions. One of them filters the list of available schemas, the other converts the output of that filter into a usable prefix which can be prepended to all datasets using that schema to generate their main table. A preview of the values selected is shown in the table below them.</p> <p>By emptying these two fields, partitioning will be turned off again.</p> <div style="border: 1px solid orange; background-color: yellow; padding: 10px; margin-top: 10px;"> <p>When two schemas are joined, if the LHS is partitioned and the RHS is not, then the join is made. If both sides are partitioned, then the join is only made if there is a partition with the same name on both sides (as determined by the second regular expression). If the LHS is not partitioned but the RHS is, then the results are unpredictable and may cause incorrect SQL to be generated.</p> </div>
Updating schemas	S	<p>Select the schema you want to update, and use the Update option in the Schema menu. If you want to update all schemas, use Update All instead.</p>
Relational cardinalities	S,AS	<p>Right-click on the relation in the schema diagram and choose the appropriate cardinality. The change will make it into a handmade relation which indicates that your choice overrides the system's choice.</p> <p>Relations which join two primary keys can only be 1:1. Relations between two foreign keys can only be M:M. Those between a primary and a foreign key can either be 1:1 or 1:M.</p> <p>1:M relations are directional. Those between a primary key and a foreign key will always have the 1 end at the primary key. Those that are between two foreign keys must specify which end is to be treated as the 1 end.</p>
Key guessing	S	<p>When a schema is first loaded from a database which does not support referential integrity, such as those using MySQL's MyISAM table engine, there will be no foreign keys or relations defined. Turning this option on in the Schema menu will make MartBuilder attempt to guess where the keys and relations might be in this case. By default it is turned on if the database provides no foreign key information of its own when the schema is first added to MartBuilder.</p>
Incorrect or unwanted keys and relations	S,AS	<p>If MartBuilder has guessed a foreign key or relation using the key guessing algorithm and has got it wrong, or if you simply don't want it to be included in the transformation, you can mark this by right-clicking on that key or relation in the diagram and selecting the Mask key/relation menu option. The key/relation will then be ignored as if it did not exist.</p>

Adding keys and relations	S,AS	<p>If MartBuilder didn't guess all the foreign keys and relations, or the primary key information was missing from some tables, then you can add them in. To add a key, right-click on the table and choose Create foreign/primary key. This opens a box with all columns of that table listed on the left. Use the arrows to move them across to the right to build the key. The key columns will be made in the order they appear in the list on the right.</p> <p>To add a relation, drag one key to another, or right click the first key and use the Create relation option to select the target key. The cardinality of the resulting relation will depend on the types of the two keys. If they are both primary, it will be 1:1, but if they are both foreign it will be M:M. All others will be 1:M.</p> <div style="border: 1px solid orange; background-color: yellow; padding: 10px; margin-top: 10px;"> <p>MartBuilder cannot create datasets with a main or subclass table which is focused on a table with a composite (multi-column) primary key. It also cannot add dimensions or subclasses to a main table based on a table with no primary key at all.</p> </div>
Editing keys	S,AS	Right-click on the key and choose the Edit key option to change the columns that appear in that key.
Removing keys and relations	S,AS	Right-click on a key or relation and use the appropriate remove option to remove it.
Creating datasets	S,AS	<p>To create a dataset, work out which table you want to focus your dataset main table on. Right-click on it and choose Create dataset, and click OK. Alternatively, use the similar menu option in the Dataset menu.</p> <p>To create partition tables or unrolled datasets, see the descriptions in the earlier section of this manual.</p>
Removing or renaming datasets	D	Remove/rename a dataset by selecting its tab then choosing the appropriate option from the Dataset menu, or right-clicking on the tab and removing it from the context menu.
Renaming datasets	AD	Double-click on the dataset name in the All Datasets tab. Edit the name. Hit Enter to accept it, Escape to cancel the change.
Replicating datasets	D	Choose the dataset tab, then use the context menu in the tab or in the Dataset menu to replicate it. Replication will create an entire copy of the dataset including all modifications you have made to it.
Extending datasets	D	Use this option to suggest further datasets which may be of use. It will create datasets based around all tables which refer to the same keys as the main and subclass tables of the existing dataset but are not included in the existing dataset.
Invisible datasets	D,AD	This option has no effect at present other than to change the displayed colour of the dataset. It will be used in future to assist at the configuration stage.
Explaining datasets	D	The dialog shown by the Explain dataset option in the Dataset menu will show all the relations and tables used by the currently selected dataset. Any changes made to this diagram will affect this dataset only.

Explaining tables	D	The dialog shown by the Explain table option in the context menu of each dataset table will show all the relations and tables used by the currently selected table. Any changes made to this diagram will affect this table only.
Renaming tables and columns	D,ET(T)	Rename a dataset table (<i>D</i> , <i>ET(T)</i>) or column (<i>ET(T) only</i>) by right-clicking on it and choosing the appropriate option. Alternatively, double-click it and type the name in directly. Hit <code>Enter</code> to accept the new name, and <code>Escape</code> to cancel the change. If the new name clashes it will be automatically made unique.
Subclassing relations	ED	Right-click on a relation to find the option to subclass it. Subclassed relations cause subclasses of the main table to appear, if they can be reached by following a chain of 1:M relations from the table upon which the main table is based.
Expression columns	D	<p>Right-click a dataset table to add an expression column to it. The column will be given a default name which you can rename later.</p> <p>The expression definition requires you to assign aliases to a number of columns in that dataset table, then use those aliases to construct a SQL expression. The SQL expression can be any valid expression for the RDBMS you will be building the mart on.</p> <p>You will also need to specify whether the expression requires a group-by clause to be used (e.g. if it is a <code>sum()</code> or <code>avg()</code> -style expression).</p>
Optimizer columns	D	<p>Optimiser columns contain counts or true/false flags indicating presence/absence of associated rows in dimensions.</p> <p>The Dataset menu has a submenu listing the various types of optimiser columns available. Only one type can be used per dataset, and will affect all tables in that dataset. It is optional whether to create indexes on those columns or not.</p> <p>See the earlier section of this manual for full details on the subject.</p>
Indexing columns	D,ET(T)	Use the index option in the context menu of a column to turn on indexing or not. This will toggle the creation of an index on that column when the mart is built.
Partitioning tables and datasets	D,ET(T)	<p>The context menu on a dataset or dataset dimension table will allow the table to be partitioned.</p> <p>See the earlier section of this manual for full details on how this works.</p> <p>Be careful not to specify too many partitions else you may end up with too many tables for your RDBMS to support. Main and subclass tables cannot be partitioned.</p>
Masking tables and columns	ED,ET(S)	Right-click a table or relation to find the option to mask it. This causes the transformation algorithm to ignore it completely.
Masking dimensions	D	Right-click a dimension to find the option to mask it. This will prevent SQL being generated for this dimension, but the definition will be retained in case you change your mind later.

Compound relations	ED,ET(S)	<p>A compound relation allows itself to be included multiple times, either by forking the transformation at the point it is reached, or by allowing a circular reference to continue past the relation even though it has already been used. The forking option is turned off by default, to turn it on use the Follow in parallel checkbox in the dialog.</p> <p>The number in the dialog is a maximum – for forking, this is the number of times it will be forked, but in other cases this is the maximum number of times it will allow itself to be used in a circular relation. If it is not reached that number of times, then the actual number of times it is included will be lower.</p> <p>If a relation is compounded which is the underlying relation for a dimension or subclass table, then this will cause that dimension or subclass table to appear multiple times.</p>
Replicating dimensions	D	This is a wrapper option for the compound relation option, and simply changes the relation underlying this dimension into a compound relation. You can choose how many times to compound it.
Recursive subclasses	D	See the earlier section of this manual on this subject.
Left/inner joins	ET(T)	MartBuilder by default assigns left or inner joins to various relations according to their context. Those which are left join have Left join ticked in the context menu for the relation. Those which are not will not have this box ticked. You can override this by ticking/unticking the box yourself.
Mask table	ET(T)	If you choose this option on a table in this view only, then it is equivalent to finding the table in the schema tab of the Explain table dialog and masking all the relations that lead to it.
Start transform from here	ET(T)	<p>Sometimes you will have a transformation that includes all rows from a large table as the first step, then in a subsequent step imposes a restriction which removes a large number of those rows, before the final steps join just those remaining few rows to some more tables. It would be more efficient if you did the restriction step first, so that all joins in the transformation only operated on the subset of rows. This can be done by choosing Start transform from here on the step which you would like to start from. The joins will then be reordered to place this step first. Note that this may cause some columns to be renamed according to the new join order (columns used to join tables are always taken from the RHS of the first join that uses them), and you will gain one extra step at the end which is a left-join to the parent main or subclass table to reinstate the complete set of keys on the dimension.</p> <p>To undo it, right-click on the step again and click the option again. The start point will revert to the default. To change to a different start point, right-click on the new start point step you want and choose the option. You don't have to undo the existing start point before changing it to a different place.</p> <p>This option is for dimensions only. It will only work if (a) you have not got any unrolled relations in your dataset table, and (b) you have not used the loopback wizard on the dataset table. If you have done either, you should undo their effects before trying to use this. The use of this option also precludes the use of unrolled relations and the loopback wizard on the same table, although you can still use standalone loopback relations.</p>

Restricting tables	ED,ET(S)	This option allows the equivalent of a SQL where-clause to be applied to a table, so that whenever it is included in a transformation only the matching rows are included. Like an expression column it requires the actual columns to be assigned aliases, then those aliases to be used in a SQL expression which is valid in the target RDBMS.
Restricting relations	ED,ET(S)	<p>This option allows extra join restrictions to be applied to a relation, so that whenever it is included in a transformation only the matching rows at the target end of the relation are included. Like an expression column it requires the actual columns at both ends to be assigned aliases, then those aliases to be used in a SQL expression which is valid in the target RDBMS.</p> <p>If you are restricting a compound relation, you will be asked which iteration of that relation the restriction should be applied to. This means that if you select 2, for instance, the restriction will only be applied the second time the relation is followed.</p>
Unrolled dimensions	D	<p>This option forces the transformation algorithm to recurse down a dimension in circles until it hits the bottom. It causes the dimension to be merged into the main or subclass table it belongs to, and for all the descendants it describes to appear as rows in the main table, as a cross product of the parent entries and all descendants.</p> <p>This option requires there to be two relations from the source main table to the table underlying the dimension - i.e. there should be two very similar dimensions in your dataset. One of them should be merged before this option is then used on the other. The merged one is the parent node, the unrolled one is the child node.</p> <p>The dialog that pops up asks for a column to use to describe each child node, which would usually be the column containing the name of that node (especially so for ontologies).</p> <p>See the earlier section of this manual for full details of how this works, including what the Reverse sense checkbox does.</p>
Forcing relations	ED,ET(S)	<p>The transformation algorithm has boundary conditions where it will stop walking relations and including them in the dataset. If you find that it stops too early, you can locate the relation it has missed and right-click it and choose the Force relation to insist that the algorithm includes it. This will only work if the relation is linked to a table which is already being included, otherwise the algorithm will never reach it.</p> <p>Forced relations always cause the tables linked by the relation to be merged into one table, and may result in further tables beyond the forced relation being included in the transformation if the algorithm conditions are met.</p>
Merging dimensions	D	Merge a dimension into the main table by using the context menu on the dimension. This in effect makes the underlying relation a 1:1 relation, causing the dimension's transformation steps to be appended to the transformation steps for the main table. The dimension itself will no longer be produced in generated SQL.

4.4 MartRunner

MartRunner automates the task of executing SQL generated by MartBuilder. It provides monitoring and email notification facilities and can tell you if, why, and when something goes wrong. It also enables you to restart the sequence of SQL commands from the point of failure after you have resolved the problem.

4.4.1 Starting and stopping MartRunner

MartRunner operates via JDBC calls and so does not need to run on the same server as your database server.

The MartRunner application is designed to be run in the background as a server daemon thread. It opens a port and listens for commands on that port. There is no further interaction with the user on the command line - all interaction is done via commands on that port. MartBuilder has a monitor window which makes such interaction very simple.

Start MartRunner on Unix/Linux by changing to the *martj* directory, and typing:

```
bin/martrunner.sh 1234
```

1234 should be replaced with the port number you wish MartRunner to listen on. This port number (and the hostname of the server you started it on) should be used in MartBuilder to connect to and communicate with MartRunner later on.

If you have special JDBC driver jar files you wish to use with MartRunner then they must be added to the classpath before starting it.

To stop MartRunner, it is best to stop any active jobs via the MartBuilder monitor dialog first and wait for them to complete. Then you can simply Ctrl-C the MartRunner process to kill it. Stopping jobs first is not necessary but removes the need for it to check for stopped jobs when it is restarted later on.

4.4.2 Sending jobs to MartRunner

When using the **Save SQL** option in MartBuilder to generate SQL, choose the Run SQL option and specify the hostname and port number that MartRunner is listening on. MartBuilder will then send the SQL to MartRunner, and open the MartRunner monitor window on completion so that you can see and start your job.

MartRunner uses the database connection, username and password details from the schema table underlying the main table of the first dataset. It uses this connection for all operations. You need to ensure that this user has read access to all incorporated schemas and has write access to the target schema.

If MartRunner has to use a different connection string to reach the database, e.g. if MartBuilder is on your local machine and using SSH tunnels, but MartRunner is on another machine that can see the database server directly, then you can specify the database server hostname and port number for MartRunner to connect to in the **Save SQL** dialog box. Leaving these blank will cause MartRunner to attempt to use the same connection parameters as MartBuilder did.

4.4.3 Monitoring MartRunner

Use the **Monitor construction progress** menu option in the **Mart** menu of MartBuilder to open the MartRunner monitor dialog.

Jobs currently defined in MartRunner are listed in the left-hand panel. Click on a job to view the details for it in the right-hand panel. Use the **Refresh** button at the bottom of the list to update the list and the details of the currently selected job. In addition these will both automatically update once every minute.

Actions performed in the monitor dialog will take immediate effect in MartRunner, but the dialog will not necessarily update immediately to reflect the new state of MartRunner. Use the **Refresh** button or wait for the next per-minute refresh cycle to see the changes reflected in the dialog. If per-minute cycles (60 seconds) are too frequent, use the number field beside the **Refresh** button to specify a longer delay between refreshes. The number is measured in seconds.

If you get **ProtocolException** errors it is most likely that communication has been lost between MartBuilder and MartRunner. Check that MartRunner is still running, and that the hostname and port details you specified in MartBuilder are correct.

4.4.3.1 Queuing / Unqueuing commands

After a new job has been created and sent to MartRunner by MartBuilder, all commands for that job will be queued by default.

MartRunner will only execute commands that have been queued (or have been stopped by unexpected termination of MartRunner mid-job – see later for details). To be queued simply means that MartRunner includes this command on the list of commands that need doing.

The order in which commands are queued does not affect the order in which those commands are executed, which is predetermined by MartRunner and not modifiable by the user.

Right-click on any part of the tree of commands in the right-hand panel to see menu options to queue or unqueue commands. If multiple nodes of the tree are selected, the menu option will apply to all selected nodes. Selecting a node which has child nodes (i.e. is not a leaf node) will have the same effect as opening that node and selecting all its children.

Queued commands show in **purple**. Unqueued commands show in black.

4.4.3.2 Reordering groups of commands

Top-level groups can be dragged and dropped around the tree to change the order in which they are executed. Top-level groups are those nodes of the tree which are children of the root node. This can be done at any stage.

4.4.3.3 Starting / Restarting / Stopping jobs

To start a job, or restart one that is showing its status as stopped, select the job in the list on the left to view its details in the right-hand panel. Then use the **Start** button to set it going. To stop the job, use the **Stop** button. A job will only actually stop when the currently executing SQL statements have completed.

Stopped commands show in **bold orange**. Running commands show in **bold italic blue**. Failed commands show in **bold red**. Successfully completed commands show in **green**.

4.4.3.4 Removing jobs

Right-click on the job in the job list and choose the **Remove** menu option.

4.4.3.5 Timing commands

Select a node of the tree in the right-hand panel to view the start and end times and duration for that node in the panel below the tree. If the 'ended' box in the panel is empty it means either that the command has not been started yet, or is currently running and has not completed or failed yet.

Nodes that represent single commands will show the time for just that command. Nodes that represent groups of commands will show the time for the whole group.

4.4.3.6 Error messages from failed commands

Select an individual command in the tree to view messages from that command in the panel below the tree. Only failed commands will have messages, which will consist of the Java stack trace raised by the failed command within the MartRunner application.

4.4.3.7 Email notifications

If you would like emails to be sent to you detailing progress of your job, select the job in the left-hand list in order to show its details on the right panel. Enter your email address in the box above the tree on the right and hit the **Update** button. Changes to email addresses take place immediately and can be made at any point before or during job execution.

4.4.3.8 Multi-tasking and threads

MartRunner knows how to break up jobs into smaller chunks that can safely be run in parallel without breaking any of the inter-command dependencies. By default it does not do this, however you can use the Threads option in the panel above the job details tree to increase or decrease the number of threads currently assigned to the job.

The threads are updated once every 5 seconds so the change will not take place immediately. If thread count is increased, new threads will be started at the next update round (assuming that the job can be broken up into small enough pieces to give one piece to each thread). If the count is decreased then at the next update round any excess threads are told to stop once they have finished their current SQL statement.

The optimal number of threads depends entirely on your database server. Too many will overload it, too few will mean that the job will not get done as fast as it could have been if more threads were assigned.

4.5 Configuring MartBuilder (optional)

MartBuilder stores all its configuration in a folder in the home directory of the user that installed it. This folder is called `.biomart` and is created the first time *MartBuilder* is run.

Within this folder is a file called `properties`. This contains a number of configuration settings that affect the appearance and behaviour of *MartBuilder* and *MartRunner* but for initial usage at least there is **no need to edit any of the values in the *properties* file.**

Setting	Value and significance
mail.from	The 'from' address for mail sent by <i>MartRunner</i> .
smtp.hostname, smtp.username, smtp.password	The SMTP server/username/password to use for mail sent by <i>MartRunner</i> . The username and password are optional.
maxthreads	The maximum number of parallel threads that <i>MartRunner</i> is allowed to start within a single job.
classCacheSize	The number of history entries to record for things like recently used files and schemas in <i>MartBuilder</i> .
maxunits	The maximum number of transformation steps that can be displayed in the Explain Table dialog in <i>MartBuilder</i> before it gives up and shows a 'too many units' message instead. This dialog is very memory- and processor-intensive so too high a value may lead to unusable response times and/or out-of-memory errors.
lookandfeel	The Java look-and-feel class to use when rendering <i>MartBuilder</i> and <i>MartRunner</i> windows.
currentOpenDir, currentSaveDir	These are internal settings for use by <i>MartBuilder</i> , which record the last directory chosen by the user to open/save a file.

5 Configuring Marts

5.1 Server-side Configuration

5.1.1 Dataset configuration file

A BioMart server is an RDBMS server of your choice that contains one or more marts.

Inside every mart is a private dataset called **meta**. This private dataset, like all other datasets, has a set of tables which conform to the BioMart conventions.

The **meta** dataset defines all the datasets available in the mart. For each dataset it makes available an XML document fully describing the dataset and how it should be presented to users. This document is the dataset configuration file.

The dataset configuration file defines which attributes and filters should be available, how they are grouped together, in what order they should appear, and what names should be used to describe them. The file also defines the way in which filters can interact with each other (for example selecting a chromosome from the chromosome filter alters the bands available in the band filter), and the different ways in which users can specify values for the filters to restrict results by.

When a client connects to the server, the first thing it does is to download these XML dataset configuration files in order to find out what is available to present to the user.

Although the file can be created and edited by hand, the easiest way is by using MartEditor.

5.1.2 MartEditor

MartEditor provides everything you need to create dataset configuration files and set up the **meta** dataset inside your mart.

5.1.2.1 Starting

On a Mac, MartEditor can be started by double-clicking the *MartEditor* application icon if you downloaded the binary distribution.

On Windows, you can double-click the *marteditor.bat* file in the *bin* folder of the *martj* distribution.

On Unix/Linux, you need to change to the folder containing your *martj* distribution and type:

```
bin/marteditor.sh
```

In all cases this should result in the display of an empty MartEditor application window. The main functions such as database connection, importing and exporting of configurations are accessed from the **File** menu on the top bar.

5.1.2.2 Connecting to a mart

The first thing you need to do is to establish a connection. MartEditor remembers your connection settings from last time you used it, so that you don't have to reconnect every time you start it.

If you have previously made a connection and that connection is not valid, you will see an error dialog during startup. This can safely be ignored but you will then need to establish a new connection.

Using the **Database connection** option from the **File** menu, bring up the connection dialog. This dialog box allows you to enter all the usual settings required for connecting to a database. Enter the details for the database that contains your mart, then hit **OK**.

Note that the values entered are case-sensitive, and that if you are using MySQL then **Database** and **Schema** should contain the same values.

You are now connected to your mart. The title bar will change to reflect the new connection settings.

5.1.2.3 Setting up a dataset

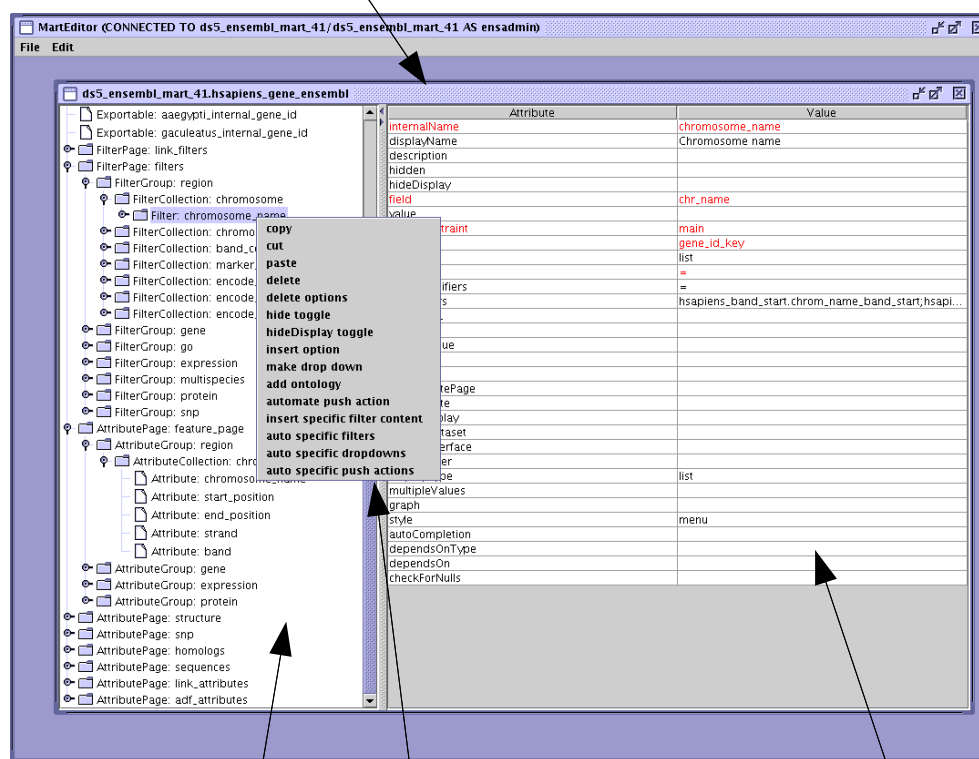
Use the **Naïve** option in the **File** menu. A dialog box will open asking you to select your dataset, following which your dataset configuration will appear on screen. To save the configuration use the **Export** option in the **File** menu. Naïve creates an attribute for each column in the dataset and a filter for each main table column with the exception of those ending **_key**, or containing all nulls, or duplicated main table columns in the subclass tables. Filters are not created automatically for dimension table columns as these can cause query performance problems so it is best to add these manually if required (see below). The exception to this is any dimension table column ending **_list** will become an external identifier filter in the naïve configuration. The final column naming convention used by the naïve generation is any main table column ending **_bool** will become a boolean filter rather than the default value filter.

5.1.2.4 Browsing an existing dataset

Use the **Import** option in the **File** menu. Use the dialog box that appears to browse and select an existing dataset configuration. The **dataset configuration** XML has a hierarchical structure consisting of **attributes** and **filters** grouped into **pages**, **groups** and **collections** which describe the layout in the graphical user interfaces. In addition, the configuration consists of **exportables** (ordered lists of attributes) and **importables** (ordered lists of attributes) used to define the linking between datasets.

Navigation through the configuration is through the **tree** interface in the **left panel** with editing of the actual XML attributes at each node in the main **right panel**. Functions specific to a particular item in the configuration tree are described in the **context menu** accessible by **right clicking (or Ctrl-clicking on Macs)** on one of the tree nodes.

Configuration file window



Context menu

Main panel

Tree panel

5.1.2.5 Editing Configuration settings

After importing as described above, MartEditor can be used to edit the configuration before exporting the changes back using the **Export** option in the **File** menu. This is typically done after creating a naïve configuration to turn off or add particular filters and attributes and alter their layout, appearance and behaviour in the various user interfaces.

The first thing you will probably want to edit after creating a naïve configuration are the **DatasetConfig** settings such as the **displayName** and **version** which are used for display purposes. You should set **visible** to '0' for utility datasets not to be displayed and **defaultDataset='true'** if you want this dataset to appear as the default.

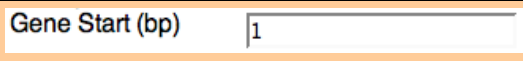

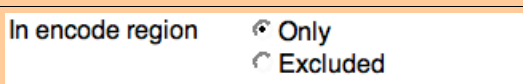
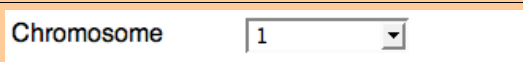
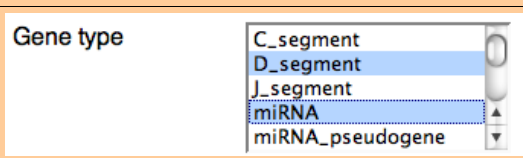
Most of the other items in the configuration have an **internalName**, **displayName**, and **description** you can edit. The **internalName** is used to identify the item in the BioMart software and should not contain spaces and should also be unique within the attribute hierarchy for attributes and unique within the filter hierarchy for filters (i.e filters and attributes can use the same **internalName** if you want). In addition there is the ability to **hide** an item which means it will be ignored by the BioMart APIs which is useful for turning off an attribute or filter that you do not want to reappear next time you update the configuration against the database again (see below). You can also **hideDisplay** an item which means it is available in the BioMart APIs for functions such as linking between datasets but is not displayed in the user interfaces. Generally you will just hide attributes

and filters you do not want after naïve configuration generation and maybe change some of the `displayNames` to more readable names.

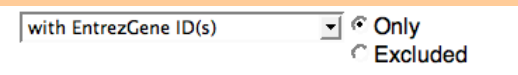

For attributes and filters, the **field** and **tableConstraint** settings describe which table and column the attribute or filter refers to. The `tableConstraint` should be main for all main or subclass tables, and for other tables it should include only the `content__type` portion of the table name. The **key** setting names the unique column on the table this attribute or filter belongs to. Usually you will not need to worry about these settings unless you are adding dimension table filters to the configuration which naïve does not do automatically. Things you may want to edit for attributes are the **linkoutURL** (everything up to the first '|' is stripped off and the %s substituted with the actual attribute value when generating hyperlinks in the interfaces) and **default** (set to 'true' if you want the attribute turned on when you first use the graphical user interface). For filters you may want to change the **qualifier** from its default of '=' to another operator such as '>'.

For filters that have predefined options, use the context menu to add those options and specify values for them. Use the **make dropdown** context menu choice to automate these based on the current set of distinct values stored in the column in this dataset. These values appear as Option items in the configuration. If another filter's values depend on the value chosen in this filter, known as a push action, use the **automate push action** context menu choice to name that other filter and link the two sets of values together based on the associations found in the current sets of values in the dataset.

Filters have a **displayType** which along with the **style** and **multipleValues** settings define how they will appear in the user interface:

displayType	style	multipleValues	Description	Screenshot
text			Text-field filter	
text		1	Text-area filter allowing IDs to be pasted/uploaded	
list	radio		Boolean filter	
list	menu		Select drop down filter	
list	menu	1	Multi-select drop down filter	

In addition filters can be dragged or copy and pasted onto a blank filter with no settings except the **internalName** and **displayType =container**. This is used to create a drop down list of boolean or text-area filters:

displayType	style	multipleValues	Description	Screenshot
text		1	Text-area filter allowing IDs to be pasted/uploaded	
list	radio		Boolean filter	

For pointer attributes and filters, everything should be blank except the **internalName**, **pointerDataset**, **pointerInterface** (should be 'default') and either **pointerAttribute** or **pointerFilter** with the name of the attribute or filter that this item should refer to. These pointer settings are used when you want to use an attribute or filter from another dataset and the BioMart software will handle all the relevant dataset linking assuming the appropriate links are defined as below.

Datasets can be linked together in queries if one dataset has attributes that another dataset can use the values from in a filter. To allow a dataset to be linked, **importables** and **exportables** must be defined. The linking can be as complex as required involving multiple attributes. An exportable defines an ordered list of attributes to be exported from a dataset. To link to another dataset an importable needs to be created on that dataset with the same **linkName** setting as the exportable. In addition if a **linkVersion** is set then it must match for the importable and exportable. The importable is created in the same way as the exportable except an ordered list of filter internalNames matching the exportable's attributes are defined.

5.1.2.6 Deleting a dataset

Use the **Delete** option in the **File** menu, and use the dialog box that appears to select the dataset configuration to remove.

5.1.2.7 Updating a dataset

If your dataset changes and you need to modify the configuration to match, then use the **Update all** option in the **File** menu to cycle through all datasets and bring them up-to-date. Changes made by this routine will automatically be exported. When you browse the dataset later, you will find that any new attributes and filters will have been grouped together in a new collection. In addition any filters and attributes that are no longer valid (because the field is missing from the new dataset or now contains all nulls) will become hidden.

5.1.2.8 Validating

If you want to check your dataset configuration for possible errors you can use the **Validate** option. This will check for various possible problems such as spaces or apostrophes in internal names.

5.1.2.9 Saving/Loading files

If you want to browse or edit the dataset configuration files by hand, you can use the **Save All** option in the **File** menu to save all of them to some location on disk. After editing them,

you can upload them again (replacing all configuration files that already exist in the mart) using the **Upload All** option in the **File** menu.

5.1.2.10 Moving to a new server

If you have set up a new version of your mart on a new server and wish to move the configuration across, use the **Move All** option in the **File** menu. Before doing this, use MartEditor to connect to your old server. Then, use the **Move All** option to connect to the new server. The configuration files will then be copied across.

5.1.3 Upgrading 0.5 to 0.6/0.7

MartEditor 0.6/0.7 can upgrade your existing 0.5 marts simply by importing and exporting them. All appropriate database and configuration transformations are handled automatically. The XML schema for 0.6 and 0.7 is essentially the same.

5.2 Client-side Configuration

Client configuration is restricted solely to registry files. Client in this sense refers to a program that connects directly to a BioMart server. It does not include users who connect via the web browser interface or web services API provided by *biomart-perl*, but it does include *biomart-perl* itself.

5.2.1 Registry Files

A registry file lists a series of locations where marts may be found.

5.2.1.1 Registry file structure

Marts and their datasets are grouped into *virtualSchemas*, within which dataset names must be unique. *virtualSchemas* appear to the end user as if all the datasets within that *virtualSchema* live in the same place.

A few example registry files are given in the *conf* folder of the *biomart-perl* distribution.

Registry descriptions within the file are contained in *RegistryDBPointer*, *RegistryURLPointer*, *MartURLLocation* and *MartDBLocation* tags.

5.2.1.2 RegistryDBPointer

These tags refer to special databases containing registry files that the client can download and parse as if the file was held locally. This allows multiple clients to share one copy of a registry file that can be updated centrally.

An example of the tag looks like this:

```
<RegistryDBPointer
  name      = "central_registry"
  host      = "martdb.ebi.ac.uk"
  port      = "3306"
  user      = "anonymous"
```

```

    password      = ""
    database      = "central_registry"
    schema        = "central_registry"
    proxy         = ""
    databaseType   = "mysql"
    includeMarts   = ""
  />

```

The proxy tag allows you to connect to the registry database via a proxy server if you need to do so.

5.2.1.3 RegistryURLPointer

This is similar to the *RegistryDBPointer* tag, except that instead of connecting to a database to retrieve the file, it connects to a URL and downloads it. The tag looks like this:

```

<RegistryURLPointer
  name      = "central_registry"
  host      = "someserver.somewhere.com"
  port      = "80"
  path      = ""
  includeMarts = ""
/>

```

5.2.1.4 MartDBLocation

This tag points directly to a mart database:

```

<MartDBLocation
  name      = "atest"
  displayName = "A Test"
  databaseType = "mysql"
  host      = "mymachine.myserver.com"
  port      = "3306"
  database   = "bloggs"
  schema     = "bloggs"
  user       = "joe"
  password   = "smith"
  visible    = "1"
  default    = ""
  includeDatasets = ""
/>

```

5.2.1.5 MartURLLocation

This tag points to a URL which responds with a dataset configuration file:

```

<MartURLLocation
  name      = "atest"
  displayName = "A Test"
  host      = "mymachine.myserver.com"
  port      = "80"
  serverVirtualSchema = ""
  visible    = "1"
  default    = ""

```

```
    path                = ""  
    includeDatasets     = ""  
/>
```

5.2.1.6 Central Registry

A central registry server is provided which contains details of the most common marts available for public use. To access this server, use the *centralRegistryDBPointer.xml* registry file in the *conf* folder of the *biomart-perl* distribution.

5.2.1.7 Where to put the Registry file

For *biomart-perl*, the registry file should live in the *conf* folder and be referenced during configuration process (see elsewhere in this document).

For MartExplorer and MartShell, the registry file is called *defaultMartRegistry* and it can be found in the *martj-0.7/data* directory.

For applications that users write using the Java or Perl API, the registry file can live anywhere as the application will specify where to look for it.

6 Querying Marts

6.1 MartShell

MartShell allows users to enter queries as commands in the Mart Query Language (MQL). Results from the queries are then printed on screen. MartShell is a console application and is designed so that it can be scripted for frequent or repetitive tasks.

MartShell is built around the BioMart Java API in the *martj* distribution. The queries it can run are therefore restricted to the queries it is possible to define using this API.

6.1.1 Starting

On a Mac, MartShell can be started by double-clicking the *MartShell* application icon if you downloaded the binary distribution.

On Windows, you can double-click the *martshell.bat* file in the *bin* folder of the *martj* distribution.

On Unix/Linux, you need to change to the folder containing your *martj* distribution and type:

```
bin/martshell.sh
```

In all cases this should result in MartShell starting up and parsing your registry file. See the section on client-side configuration elsewhere in this document for information on where to put the registry file and how to configure it. MartShell will use the default registry file that comes with it to connect to the central BioMart servers. You can edit this file as required.

You should see the following on-screen once MartShell has successfully started:

```
Starting Interactive MartShell
```

```
MartShell: An Interactive User Interface to BioMart databases  
based on Mart Query Language (MQL)  
type 'help' for a list of available commands, or type 'help  
command' to get help for a particular command.
```

```
MartShell>
```

The MartShell prompt, **MartShell>**, indicates that MartShell is ready to be given a command.

6.1.2 Using

Using MartShell involves typing MQL commands into the prompt. Each command ends with a semi-colon (;), and you must hit **Enter** to submit the command for processing.

A summary of MQL commands follows later in this document. To exit MartShell at any stage, issue the **exit** command.

MartShell allows tab-completion of dataset, filter and attribute names. You can start typing the first few letters of the name, then hit **Tab** to see matching names and select one. It also supports the use of the up- and down-arrow keys to scroll through the history of commands issued.

Most MQL commands require a dataset to be selected before they can run. To select a dataset, use the **use** command to specify the name of the mart the dataset is in, and the name of the dataset itself. Once set, the dataset will remain selected until another **use** command is issued or MartShell exits. In this example we are selecting the *hsapiens_gene_ensembl* dataset in the *ensembl* mart:

```
use ensembl.hsapiens_gene_ensembl;
```

Alternatively, each individual MQL command can be prefixed with a **using** clause. For example to retrieve some human gene data:

```
using ensembl.hsapiens_gene_ensembl get ensembl_gene_id where  
chromosome_name=1 ;
```

The best way to learn MQL is by example. A series of MQL example queries are included with MartShell. To view them, type:

```
help EXAMPLES;
```

6.1.3 Batch jobs and scripting

MartShell's biggest advantage over the other BioMart applications is that it is command-line based and can be incorporated into batch jobs or scripts. Simple text files containing one or more MQL commands can be piped into it, and the query results redirected to an output file.

This section describes scripting and batch jobs for Linux/Unix users. Users of other operating systems will need to modify these examples.

There are two methods of scripting MartShell. One method is to execute scripts from within MartShell. These scripts are text files containing one or more MQL statement:

```
execute script "/home/biomart/myexample.mql";
```

The other method is to pass the commands into MartShell on startup. If you do this, then you must remember to include an **exit** command so that MartShell exits after executing the script:

```
bin/martshell.sh < /home/biomart/myexample.mql
```

Output from the query will be written to standard output (STDOUT), unless MartShell is explicitly told to write it to a particular file. To do this, issue the **set** command in your MQL before issuing any **query** commands:

```
set output file='/home/biomart/example/output.txt';
```

To stop writing to the file and restart output to standard output, issue the **unset** command:

```
unset output file;
```

6.1.4 MQL guide

6.1.4.1 environment

```
environment [<type> [<parameters>]];

type: mart|output|dataset|datasetconfig
parameters: format|separator|file
```

Displays the current environment settings. The **parameters** are only required when the **type** is **dataset**.

6.1.4.2 set

```
set <type> <value>;

type: mart|dataset|prompt|output|verbose
value: <name>|<prompt>|<key>=<value>|on|off
prompt: any valid quoted string
key: file|format|separator
name: any valid mart or dataset name
value: <filename>|<separator>|<format>
filename: any valid quoted string
separator: any valid quoted string
format: tabulated|fasta
```

Sets an environment setting. The **mart** and **dataset** types require a mart or dataset **name**. The **prompt** type requires the **prompt** value. The **verbose** type requires an **on** or **off** value. The **output** type requires one of three possible **key=value** combinations. Specifying a **filename** that begins with >> indicates that the file is to be appended to rather than overwritten.

6.1.4.3 unset

```
unset <type> [<value>];

type: mart|dataset|prompt|output|verbose
value: <key>
key: file|format|separator
```

Unsets an environment setting. For **prompt**, this resets it to **MartShell>**. For **verbose**, it sets it to off. For **output**, it sets **file** to standard out, **format** to tabulated, and **separator** to a single tab character.

6.1.4.4 use

```
use [<mart>.<dataset>[.<interface>]];

mart: any valid mart name
dataset: any valid dataset name
interface: any valid interface name
```

After issuing this command, all subsequent **get** commands will use this dataset.

6.1.4.5 *list*

```
list <type>;

type: datasets|datasetconfigs|filters|attributes|procedures|marts
```

Lists the names of all known objects of the given type.

6.1.4.6 *describe*

```
describe <type> <parameters>;

type: dataset|mart|filter|attribute|procedure
parameters: any valid dataset, mart, filter, attribute or
procedure name
```

Describes the named object of the given type.

6.1.4.7 *update*

```
update <type> <parameters>;

type: dataset|datasets
parameters: <datasetname>|[from <martname>]
datasetname: any valid dataset name
martname: any valid mart name
```

Reloads the configuration for a given dataset, or all the given datasets in a particular mart. If no parameters are given for the **datasets** type then all datasets are updated. The parameters are compulsory for the **dataset** type.

6.1.4.8 *get*

```
[count_focus_from] [using <mart>.<dataset>] get <attribute_list>
[where <filter_list>] [as <procedure>];

mart: any valid mart name
dataset: any valid dataset name
attribute_list: <attribute>[,<attribute_list>]
attribute: any valid attribute name
filter_list: <filter>[ and <filter_list>]
filter: <filtername> {excluded|only|in <value_list>}|{<|=|>|>=|
<|<=}<value>}
filtername: any valid filter name
value_list: <path>|(<comma_list>)|<procedure>
comma_list: <value>[,<comma_list>]
value: any string
path: any valid filename or URL
procedure: any valid stored procedure name
```

Executes a query. The syntax is very similar to that of SQL. The dataset used is the currently selected one (see **use**) unless the **using** clause is included. The **count_focus_from** clause makes the query count matching entries in the main table rather than return any results.

6.1.4.9 execute

```
execute <type> <parameter>;
```

type: history|procedure|script

parameter: <procedure>|<script>|<range>

procedure: any valid named procedure name

script: any valid filename or URL pointing to an MQL script

range: (refer to the history command)

Executes commands either from history, or from a named procedure, or from file. The **range** parameter is only for the **history** type. The **procedure** parameter is for the **procedure** type, and the **script** parameter is for the **script** type.

By specifying the **as** clause, the query will not be executed but will be stored as a stored procedure with the given name.

6.1.4.10 saveToScript

```
saveToScript <range> <filename>;
```

range: (refer to the history command)

filename: the name of the script file to create

Saves the specified lines from the history buffer as a file.

6.1.4.11 loadScript

```
loadScript <filename>;
```

filename: the file containing MQL commands to load

Loads the specified file of MQL commands and appends it to the end of the history buffer. Commands are loaded but are not executed.

6.1.4.12 history

```
history [<range>;
```

range: <n>|<n>,|<n>,<y>|,<y>

n: the first line in the history buffer to include, defaults to 1

y: the last line in the history buffer to include, defaults to the last line in the buffer

Displays the contents of the history buffer, which stores all MQL commands entered into MartShell.

6.1.4.13 add

```
add <type> <parameters>;

type: mart|dataset|datasetconfig
parameters: <martparams>|<datasetparams>|<dsconfigparams>
martparams: <key>=<value>[ <martparams>]
key: host|port|user|password|instanceName|databaseType|jdbcDriver
value: any valid quoted string
datasetparams: from <path>
dsconfigparams: from <path> [as <interface>]
path: any valid filename or URL
interface: any valid string
```

Adds a mart, dataset or datasetconfig to the session and sets it as the current default. For marts, if **databaseType** is not specified it defaults to **mysql**. **jdbcDriver** defaults to the MySQL driver class that ships with *martj*. The **interface** setting for **datasetconfigs** defines the name of the interface that this config will represent.

6.1.4.14 remove

```
remove <type> <parameters>;

type: mart|datasets|dataset|datasetconfig|procedure
parameters: <name>|from <name>
name: any valid mart, dataset, datasetconfig or procedure name
```

Removes the named object from the current session. The **from** syntax is for removing datasets from a specified mart and the **name** in this case should be that of the mart.

6.2 MartExplorer

MartExplorer is a simple GUI application which allows users to construct queries and browse the results. It is designed for quick simple queries rather than larger more detailed queries, for which it would be better to use MartView.

MartExplorer is built around the BioMart Java API in the *martj* distribution. The queries it can run are therefore restricted to the queries it is possible to define using this API.

6.2.1 Starting

On a Mac, MartExplorer can be started by double-clicking the *MartExplorer* application icon if you downloaded the binary distribution.

On Windows, you can double-click the *martexplorer.bat* file in the *bin* folder of the *martj* distribution.

On Unix/Linux, you need to change to the folder containing your *martj* distribution and type:

```
bin/martexplorer.sh
```

In all cases this should result in MartExplorer starting up and parsing your registry file. See the section on client-side configuration elsewhere in this document for information on where to put the registry file and how to configure it. MartExplorer will use the default registry file that comes with it to connect to the central BioMart servers. You can edit this file as required.

6.2.2 Building queries

MartExplorer allows the user to design a number of queries in parallel. Each query has its own tab in the tabbed portion of the window. A new query can be added by using the left-most button in the button bar, with an icon of a blank sheet of white paper.

The datasets available are read from the user's registry XML file. Use the **Add Mart** option in the **Settings** menu to add further datasets, in a similar manner to the MQL **add** command. The **Reset** option in this same menu will delete all existing queries, remove all added marts, and reset MartExplorer to its initial startup state.

6.2.2.1 Defining

In the query area of the MartExplorer window there is an empty field labelled **Dataset**, with a button labelled **Change**. Click the **Change** button to set the dataset for this query. The dataset field will disappear and a new tabbed panel will appear with one tab for each page of attributes available in this dataset. This is the attributes tab, and can be returned to at any time by clicking on the **Attributes** label in the tree to the left of the window.

Attributes are arranged into groups, with one tab for each group down the left-hand side of the attribute page. Select attributes for your query by ticking the boxes beside them. Attributes can be selected from multiple groups in one page, but only attributes from a single attribute page can be included in any single query.

As you select attributes their names will appear in the tree on the left of the window. Clicking on an attribute in this tree will return you to the group and page for that attribute. Attributes can be removed by unticking the box beside them.

Filters are organised along similar lines as attributes and work in the same way, and can be accessed by clicking the **Filters** label in the tree on the left.

Once you have selected your attributes and filters, click on the **Format** label in the tree on the left. Here you can choose which output format you want your results to appear in. The FASTA option will only be available if you have selected an appropriate sequence attribute from the dataset.

6.2.2.2 Executing

There are two buttons on the button bar which allow execution of queries.

The first one has an icon of a small green triangle with a green letter F. This is the count button. Clicking this button will display the number of matching rows in the main table of the dataset in the results area at the bottom of the window.

The second one has a larger icon of a green triangle. This will execute the query and display the results in the requested format in the results area at the bottom of the window.

The output of the count or query will always remove any existing output in the results area.

If a count or a query takes too long, click on the button in the button bar with an icon of a large red square to halt it.

6.2.2.3 Saving results

In the **File** menu, there is an option called **Save Results**. Using this option will save the current contents of the results area to a file.

If you have not previously saved results from MartExplorer in this session then it will ask you where to save the file. Otherwise, it will overwrite the file you specified last time.

If you wish to save the results to a different file to last time, use the **Save Results As** option in the **File** menu.

6.3 MartView

MartView is the web browser interface through which the majority of users will interact with BioMart datasets. It is designed to be as simple to use as possible.

Wherever MartView is available, MartService will also be available, However, DAS Annotation Server is available only if configured. See the section in this document about the DAS and web services API for details on how to use DAS and Webservices.

6.3.1 Web browser interface

MartView is accessed via a web browser. The URL of your own organisation's copy will depend on your system administrators, but the URL of the central MartView using the central BioMart databases is <http://www.biomart.org/biomart/martview>.

A new query can be started at any time by using the **New Query** button at the top-left.

The first task is always to select a dataset to query. Drop-down menus that allow you to select one are provided on the first page you see. You can return to this page to select a different dataset (and therefore start a new query) by clicking on the **Dataset** entry at the top of the tree on the left of the page.

After selecting a dataset, click on the **Attributes** and **Filters** entries in the tree on the left to choose the attributes and filters for your query. You can do this in any order. As you select attributes and filters they will be added to the tree. They will appear in the order you select them, and this same order will be used to organise the results later.

Wherever you see a **Browse...** button beside a filter, you can upload a plain text file containing values to use in the filter. This file should contain one value on each line of the file, with no blank lines.

You can use the **XML** button at any time to see what your query looks like in the web services API query XML format. Alternatively you can use the **Perl** button to generate a BioMart perl API script of the same query. Similarly, **URL** button features the equivalent query representation of URL Access.

If you need to know how many rows in the main table of the dataset match your filters so far, use the **Count** button. To get results, use the **Results** button.

The results shown are the first 10 results, known as preview results, and are displayed in HTML format. If you wish to change the number of preview results shown or show them in a different format, use the top two drop-down menus above them. Depending on the data model and which attributes have been chosen you may have redundancy in the output. You can eliminate these by checking unique results but the response time may increase due to a computationally intensive hashing algorithm.

Once you are happy with your results you can select the options to download the whole set to file at the top of the page and click **Go**. Again there are options to change the format and whether to make the results unique. You can select a compressed file output and if the administrator for the server you are using allows it, you may also see output options for web files. This means that the query will run in the background to be downloaded later. MartView will ask for your email address so that it can notify you when the results are ready. MartView will include a URL in the notification email that allows you to download the query results.

6.3.2 Web services interface

See the section in this document on the web services API.

6.4 Perl API

6.4.1 Prerequisites

You need to install the biomart-perl package and create yourself a registry file as detailed in section 2.4.

6.4.2 Examples

The Perl API is best explained using examples. For examples of perl API scripts, construct a query in MartView then use the **Perl** button to show the script, or see the examples below:

The first example shows you how to discover what datasets, attributes and filters are available from a given registry file:

```
my $initializer = BioMart::Initializer->new(
    'registryFile'=>'/path/to/my/registryFile.xml');
my $registry = $initializer->getRegistry;
# Schemas.
foreach my $schema (
    @{$registry->getAllVirtualSchemas()}) {

    # Marts
    foreach my $mart (
        @{$schema->getAllMarts()}) {

        # Datasets
        foreach my $dataset (
            @{$mart->getAllDatasets()}) {
```

```

# Configuration trees (interfaces)
foreach my $configurationTree (
  @{$dataset->getAllConfigurationTrees()} ){

  # Attribute trees (pages)
  foreach my $attributeTree (
    @{$configurationTree->getAllAttributeTrees()} ){

    # Attribute groups
    foreach my $group(
      @{$attributeTree->getAllAttributeGroups()} ){

      # Attribute collections
      foreach my $collection (
        @{$group->getAllCollections()} ){

        # Attributes
        foreach my $attribute (
          @{$collection->getAllAttributes()} ){

          # Print out each attribute we find.
          print "mart: ", $mart->name,
            "\tdataset: ", $dataset->name,
            "\tattribute: ", $attribute->name, "\n";

        }
      }
    }
  }
}

```

Note how the loops are nested. If you weren't interested in attributes and filters, you could just stop nesting at the dataset level and work from there. This applies to any of the levels of detail shown. Similar methods exist for filters, importables, and exportables.

The second example shows how to construct a query. The attributes in the results will be returned first in the order of the datasets added to the query, then by the order which the attributes were added to each dataset:

```

my $initializer = BioMart::Initializer->new(
  'registryFile'=>'/path/to/my/registryFile.xml'
);
my $registry = $initializer->getRegistry;

my $query = BioMart::Query->new(
  'registry'=>$registry,
  'virtualSchemaName'=>'default'
);

$query->setDataset("hsapiens_gene_ensembl");
$query->addFilter("chromosome_name", ["22"]);
$query->addAttribute("ensembl_transcript_id");
$query->addAttribute("chromosome_name");

my $query_runner = BioMart::QueryRunner->new();
# to obtain unique rows only

```

```
# $query_runner->uniqueRowsOnly(1);
$query_runner->execute($query);
$query_runner->printHeader();
$query_runner->printResults();
$query_runner->printFooter();
```

Note how adding a second dataset to the query is as simple as adding a second call to *setDataset*.

If you need to know how many rows in the main table of the dataset match your filters so far, you need to set the count flag on the query before executing the query and then use the *getCount* method:

```
$query->count(1);
$query_runner->execute($query);
print "COUNT:". $query_runner->getCount();
```

The default output format is tab-separated values (TSV). If you wish to change this output format, you need to add an extra call before creating the QueryRunner object:

```
$query->formatter('HTML');
```

The valid settings for the formatter are:

ADF, AXT, AXTPLUS, CSV, FASTA, FASTACDNA, GFF, HTML, MAF, MFA, MFASTA, TSV, TXT, XLS

For the various FASTA formats, the first attribute returned in each dataset will be used to form the sequence portion. The remaining attributes will be listed in the header for each sequence. If the query involves multiple datasets, then one sequence per dataset will be present in each row returned.

6.4.3 Extra functions

The initialiser object will download all the dataset configuration files for each mart specified in the registry file the first time it is called. On subsequent calls, cached local copies of those files are used instead. If you wish to update or rebuild your cached copy, you need to add extra flags to the initialiser object when it is created.

To update the local cache without re-downloading dataset configurations that have not changed:

```
my $initializer = BioMart::Initializer->new(
    'registryFile'=>'/path/to/my/registryFile.xml',
    'action'=>'update'
);
```

To re-download all dataset configurations:

```
my $initializer = BioMart::Initializer->new(
    'registryFile'=>'/path/to/my/registryFile.xml',
    'action'=>'clean'
);
```

The initialiser object will load all dataset configurations into memory and they will remain there until your program exits. To change this behaviour you need to pass another setting to the initialiser object:

```
my $initializer = BioMart::Initializer->new(
    'registryFile'=>'/path/to/my/registryFile.xml',
    'mode'=>'LAZYLOAD'
);
```

Specifying `MEMORY` will load all dataset configurations at startup, which is the default setting. Specifying `LAZYLOAD` will only load the ones that are actually used.

If you are using the Perl API from the same copy of *biomart-perl* as a MartView installation, the configuration files will be shared and that any action you take to modify the configuration files using one application will directly affect the other.

6.5 Java API

The Java API is out-of-date and is due for a complete rewrite. The details here are presented for sake of completeness.

Several features that are present in the Perl API and web services API are not supported by the Java API. These include placeholders and sequences.

6.5.1 Prerequisites

You will need to download and install the *martj* package, and create a registry file. Both of these steps are detailed elsewhere in this document.

6.5.2 Examples

The easiest way to explain the Java API is to walk through an example, step-by-step.

First, we need to load the registry file:

```
URL confURL = null;
try {
    confURL =
        InputSourceUtil.getURLForString(
            "/path/to/myRegistryFile.xml"
        );
} catch (MalformedURLException e) {
    e.printStackTrace();
}

RegistryDSConfigAdaptor adaptor =
    new RegistryDSConfigAdaptor(
        confURL, false, false, false
    );
```

The three `false` parameters refer to advanced settings that can be discovered by reading the JavaDocs for *martj*.

Then, we need to load the dataset config for the dataset we wish to query. Apart from a dataset name, the method also accepts the name of a virtual schema in which the dataset lives, in this case `default`:

```
DatasetConfig config =
    adaptor.getDatasetConfigByDatasetInternalName(
        "hsapiens_gene_ensembl",
        "default"
    );
```


A query object now needs to be set up and initialised using values from the dataset object that allow it to reference that dataset and work with it:

```
Query query = new Query();
query.setDataSource(config.getAdaptor().getDataSource());
query.setMainTables(config.getStarBases());
query.setPrimaryKeys(config.getPrimaryKeys());
```

Attributes are added to the query like this:

```
AttributeDescription adesc =
    config.getAttributeDescriptionByInternalName(
        "gene_stable_id"
    );
query.addAttribute(
    new FieldAttribute(
        adesc.getField(),
        adesc.getTableConstraint(),
        adesc.getKey()
    ) );
```

Filters have to be added in one of three different ways depending on what kind of filter they are. Basic filters just have a single text value associated with the filter. Boolean filters accept a null/not-null flag. ID list filters accept multiple text values.

In all three cases, the filter needs to be loaded first:

```
FilterDescription fdesc =
    config.getFilterDescriptionByInternalName(
        "chr_name"
    );
```

Then, it needs to be configured according to the type of value to be associated with the filter.

Basic filters work like this:

```
query.addFilter(
    new BasicFilter(
        fdesc.getField(name),
        fdesc.getTableConstraint(name),
        fdesc.getKey(name),
        "=",
        "22"
    )
);
```

Boolean filters work in two ways. For numeric columns, use the `isNULL_NUM` and `isNotNULL_NUM` settings. For non-numeric columns use `isNULL` and `isNotNull`:

```
query.addFilter(
    new BooleanFilter(
        fdesc.getField(name),
        fdesc.getTableConstraint(name),
        fdesc.getKey(name),
        BooleanFilter.isNULL
    )
);
```

ID list filters work like this:

```
String[] ids = new String[] {
    "ENSG00000146556.4",
    "ENSG00000197194.1",
    "ENSG00000197490.1",
```

```

        "ENSG00000177693.1"
    };
    query.addFilter(
        new IDListFilter(
            fdesc.getField(name),
            fdesc.getTableConstraint(name),
            fdesc.getKey(name),
            ids
        )
    );

```

Once your query has been constructed, you can pass it to an *Engine* object and specify an *OutputStream* instance to receive the results:

```

Engine engine = new Engine();
engine.execute(
    query,
    new FormatSpec(
        FormatSpec.TABULATED,
        "\t"
    ),
    System.out
);

```

If you require FASTA output, change the *FormatSpec* reference to `FormatSpec.FASTA`. This will use the first attribute returned as the sequence data, and all other attributes will be made into the header line of the sequence.

6.6 Web Services API - RESTful Access

The REST style web services API of BioMart is also known as MartService. It is available as a part of the MartView application. A usage summary is available at:

<http://www.biomart.org/martservice.html>

The MartService API is accessed via HTTP requests to the *martservice* script, running in the same location as the *martview* script which provides the web browser interface. As an example, if you access MartView using this URL...

```
http://www.mycompany.com/scripts/biomart/martview
```

...then you would access the MartService using this URL:

```
http://www.mycompany.com/scripts/biomart/martservice
```

The URL of the BioMart Central Portal's MartService is:

```
http://www.biomart.org/biomart/martservice
```

Overview information (metadata) about the services available is obtained via GET requests, whilst queries are submitted using POST requests.

6.6.1 Metadata

A number of metadata queries are available, which can be retrieved by appending parameters to the end of the URL. The first parameter should be appended using the ? symbol, and subsequent parameters using the & symbol, e.g.:

```
.../martservice?type=attributes&dataset=hsapiens_gene_ensembl
```

6.6.1.1 Registry file

Returns the registry file used to configure the MartService installation.

<i>type</i>	registry
-------------	----------

6.6.1.2 Datasets available

Lists out the names of available datasets.

<i>type</i>	datasets
<i>virtualschema</i>	The name of the virtual schema that the dataset lives in. (Optional, default setting is default)
<i>mart</i>	The name of the mart to display datasets from.
	The name of the mart user the dataset is owned by. (Optional, default setting is default)

6.6.1.3 Dataset configuration file contents

Displays the dataset configuration file for the specified dataset.

<i>type</i>	configuration
<i>virtualschema</i>	The name of the virtual schema that the dataset lives in. (Optional, default setting is default)
<i>dataset</i>	The name of the dataset to use.
<i>interface</i>	The name of the interface that the dataset uses. (Optional, default setting is default)
	The name of the mart user the dataset is owned by. (Optional, default setting is default)

6.6.1.4 Attributes available

Lists out the names of attributes available on the specified dataset.

<i>type</i>	attributes
<i>virtualschema</i>	The name of the virtual schema that the dataset lives in. (Optional, default setting is default)
<i>dataset</i>	The name of the dataset to use.
<i>interface</i>	The name of the interface that the dataset uses. (Optional, default setting is default)
	The name of the mart user the dataset is owned by. (Optional, default setting is default)

6.6.1.5 Filters available

Lists out the names of filters available on the specified dataset.

<i>type</i>	<i>filters</i>
<i>virtualschema</i>	The name of the virtual schema that the dataset lives in. (Optional, default setting is <code>default</code>)
<i>dataset</i>	The name of the dataset to use.
<i>interface</i>	The name of the interface that the dataset uses. (Optional, default setting is <code>default</code>)
	The name of the mart user the dataset is owned by. (Optional, default setting is <code>default</code>)

6.6.2 Queries

Queries are submitted using POST. Within the body of the request should be a single parameter called *query*. The value attached to this parameter should be a query XML document. The results will be returned in the format requested in the query XML.

6.6.2.1 Query XML syntax

This example query XML document specifies two attributes and one filter within a single dataset:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Query>
<Query
  virtualSchemaName      = "default"
  uniqueRows             = ""
  count                  = ""
  datasetConfigVersion   = "0.7" >

  <Dataset name = "mytest" interface = "default" >
    <Attribute name = "ensembl_transcript_id" />
    <Attribute name = "chromosome_name" />
    <Filter name = "chromosome_name" value= "22"/>
  </Dataset>
</Query>
```

The *Query* tag defines the query itself. The *datasetConfigVersion* setting of `0.7` indicates that you are using XML compatible with the dataset configuration being queried.

There can be multiple *Attribute* and *Filter* tags, each indicating the *name* of an attribute or filter to apply. *Filter* tags also accept a *value* to apply to the filter.

If the query uses a second dataset, then you should include a second *Dataset* tag after the first one.

Attributes in the results returned will be ordered first by the order of the *Dataset* tags in which they appear, then by the order of the *Attribute* tags within the *Dataset* tags.

The default output format for results is tab-separated values (TSV). You can change this by adding a *formatter* setting to the *Query* tag and specifying one of the format values accepted by the Perl API (see elsewhere in this document).

If *uniqueRows* is set to `1`, only distinct rows of the results set are returned.

If *count* is set to 1, then no results will be returned beyond a single number indicating how many entries in the dataset would be matched by the supplied filters.

For examples of query XML documents, construct a query in MartView then use the **XML** button to show the query XML.

6.7 Web Services API - SOAP Access

The SOAP based web services API, hereafter referred as MartServiceSoap, is also available as a part of the MartView application.

The MartServiceSoap is functionally equivalent to that of MartService, however, the access style is strictly SOAP based. The description of the service is also available as WSDL's document/literal (wrapped) format making it WS-I compliant. If you access MartView using this URL...

<http://www.mycompany.com/scripts/biomart/martview>

...then you would access the MartServiceSoap endpoint here:

<http://www.mycompany.com/scripts/biomart/martsoap>

...and the WSDL document with XSD can be accessed using the following URLs:

<http://www.mycompany.com/scripts/biomart/martwsdl>

<http://www.mycompany.com/scripts/biomart/martxsd>

The URLs of the BioMart Central Portal's MartServiceSoap endpoint, WSDL and XSD are:

<http://www.biomart.org/biomart/martsoap>

<http://www.biomart.org/biomart/martwsdl>

<http://www.biomart.org/biomart/martxsd>

There are 5 operations in MartServiceSoap (for a working example, see the *martwsdl* of BioMart Central Portal). A brief description and signature of these operations are as follows:

Operation	Input	Output	Fault	Description
getRegistry	N.A	List of all the marts	BioMart Exception message	This request returns information about all the marts hosted on a given BioMart web server. This includes martName, web server's original location, port, virtualSchema etc etc. <i>see WSDL for details.</i>
getDatasets	martName	List of all the datasets in a mart	BioMart Exception message	This request returns information such as datasetName, type, version etc etc about all the datasets in a given mart. <i>see WSDL for details.</i>
getFilters	datasetName	List of all the filters in a dataset	BioMart Exception message	This request returns names and properties of all the filters in a dataset. Optional input <i>virtualSchema</i> may be required to avoid namespace conflict. <i>see WSDL for details.</i>
getAttributes	datasetName	List of all the attributes in a dataset	BioMart Exception message	This request returns names and properties of all the attributes in a dataset. Optional input <i>virtualSchema</i> may be required to avoid namespace conflict. <i>see WSDL for details.</i>

Operation	Input	Output	Fault	Description
Query	VirtualSchemaName, header, count, uniqueRows, Dataset/s, Filter/s, Attribute/s	Retrieves count or results	BioMart Exception message	This request returns results or count. If results are requested, the input attributes take the form of output elements. Hence, the name of output elements are strictly dependent on the input. <i>see WSDL for details.</i>

6.7.1 Semantic Annotations (WSDL-S)

The MartServiceSoap also features semantic markup for attributes and filters. The process is completely automated and only requires the list of attributes and filters with the URIs of their semantic entities. The URI may point to an entry in an ontology file/location. This information can be specified through section **[semanticAnnotations]** in file **settings.conf** under **conf** directory of **biomart-perl**.

By Adding semantic annotations to the attributes and filters, the *getAttributes*, *getFilters*, and *Query* operations would return *modelReference* property containing the definition URI. This feature not only offers meaningful interaction with BioMart databases, it also increases interoperability with other databases and tools in a particular knowledge domain.

6.8 MartView URL Requests

MartView interface can be populated using URL Request containing information about the query. Once Martview is installed this feature can be used by simply pointing your browser to the following URL

```
http://<host>:<port>/<location>/martview?<URL_REQUEST>
whereby URL_REQUEST should look like

?VIRTUALSCHEMANAME=<VirtualSchemaName>
&ATTRIBUTES=<datasetInternalName>.<Interface>.<AttributePageInternalName>.<AttributeInternalName>."<OPTIONAL: comma separated list of values if using AttributeFilters>"
&FILTERS=<datasetInternalName>.<Interface>.<FilterPageInternalName>.<FilterInternalName>."<comma separated list of filter values>"
&VISIBLEPANEL=<mainpanel, attributepanel, filterpanel, linkpanel, linkattributepanel, linkfilterpanel, results>
```

Examples:

```
1-
http://www.biomart.org/biomart/martview?
VIRTUALSCHEMANAME=default&ATTRIBUTES=hsapiens_gene_ensembl.default
.feature_page.strand|
hsapiens_gene_ensembl.default.feature_page.ensembl_gene_id&FILTERS
=hsapiens_gene_ensembl.default.filters.chromosome_name."4"&VISIBLE
PANEL=results

2-
http://www.biomart.org/biomart/martview?
VIRTUALSCHEMANAME=default&ATTRIBUTES=hsapiens_gene_ensembl.default
.feature_page.strand|
```

```
hsapiens_gene_ensembl.default.feature_page.ensembl_gene_id&FILTERS=hsapiens_gene_ensembl.default.filters.biotype."miRNA,Mt_rRNA"&VISIBLEPANEL=mainpanel
```

3-

```
http://www.biomart.org/biomart/martview?
VIRTUALSCHEMANAME=default&ATTRIBUTES=msd.default.feature_page.assembly_code|hsapiens_gene_ensembl.default.structure.struct_biotype|msd.default.feature_page.pdb_id|hsapiens_gene_ensembl.default.structure.exon_stable_id|hsapiens_gene_ensembl.default.structure.gene_stable_id|msd.default.feature_page.experiment_type&FILTERS=msd.default.filters.experiment_type."Electron diffraction,Electron microscopy"&VISIBLEPANEL=attributepanel&VISIBLEPANEL=linkpanel
```

For examples of URL Access, construct a query in MartView then use the **URL** button to show the access URI.

6.9 MartView XML Requests

MartView interface can be populated using xml Request containing information about the query. This request returns complete HTML page. Once MartView is installed this feature can be used by simply pointing your browser to the following URL

```
http://<host>:<port>/<location>/martview?query=<XML_QUERY>
whereby XML_QUERY represents XML query of MartService.
```

MartView URL/XML Requests can be used to save/bookmark queries. In addition, this feature is very useful for writing canned queries having link to MartView from an external website/interface.

6.10 DAS Server

BioMart can be used as a DAS 1.5 Annotation server. Examples using the Ensembl browser as a DAS client are shown below for a BioMart served DAS source called 'hsapiens_gene_ensembl':

BioMart 0.7 Documentation

The top screenshot displays the 'Ensembl Gene Report for ENSG00000073910'. It shows a list of transcripts and their protein products. The bottom screenshot displays the 'Ensembl ContigView' page for Chromosome 13, showing a genomic track with various annotations including gene models and protein-coding regions.

A list of the sources available from our central server is available at <http://www.biomart.org/biomart/das/dsn>. This server currently returns annotation across a segment. Possible segment values could be a Feature ID or a genomic region defined by chromosome:start,end where start and end are optional e.g

http://www.biomart.org/biomart/das/default_hsapiens_gene_ensembl_ensembl_das_gene/features?segment=ENSG00000184895

http://www.biomart.org/biomart/das/default_hsapiens_gene_ensembl_ensembl_das_chr/features?segment=X

http://www.biomart.org/biomart/das/default_hsapiens_gene_ensembl_ensembl_das_chr/features?segment=13:31787617,31871805

If you wish to set up DAS Annotation Server for your own datasets you need to add a pair of Exportable/Importables to the configuration XML depending upon the type of features you would like to present, and install MartView. The following are the Exportable/Importable pairs:

- (i) Exportable/Importable pair with type = 'dasChr' OR 'dasRegionGene'. The Importable should use a set of filters for chromosome, start and end of a region. The Exportable requires gene structure data in the following order: gene ID, gene chromosome start, gene chromosome end, transcript ID, transcript chromosome start, transcript chromosome end, strand, exon ID, exon chromosome start and exon chromosome end.

- (ii) Exportable/Importable pair with type='dasRegionFeature'. The Importable should use a set of filters for region name, start and end of a region. The Exportable requires feature ID, start, end and strand.
- (iii) Exportable/Importable pair with type='dasGene'. The Importable should use a feature ID filter. The Exportable needs to define feature annotation data: feature ID, feature annotation and feature description.

Working examples of these can be seen in the 'gene_ensembl' dataset of `ensembl_mart` for all versions from `ensembl_mart_45` onwards. It is possible to just define one pair. e.g. the `dasGene` pair, to allow the server to only serve Gene DAS annotation.

7 Third-party Software

7.1 *biomaRt*

biomaRt is a Bioconductor package that provides an API in R to query BioMart databases such as Ensembl. Two sets of functions are currently implemented.

A first set of functions aims to mimic functionality of other BioMart APIs such as Martshell, Martview, etc. These functions are very general, and can be used with any BioMart system. They allow retrieval of all information that other BioMart APIs provide.

A second set of functions are tailored towards Ensembl and are a set of commonly used queries in microarray data analysis. With these two sets of functions, one can for example annotate the features on your array with the latest annotations starting from identifiers such as affy ids, locuslink, RefSeq, entrezgene, etc. Annotation includes gene names, GO, OMIM annotation, etc. The package also provides homology mappings between these identifiers across all species present in Ensembl. Genes can be pre-selected such that they fulfill a certain requirement e.g. give all human refseq ids of genes known to be involved in diabetes.

On top of this, biomaRt enables you to retrieve any type of information available from the BioMart databases from R.

An example of how to retrieve Ensembl gene IDs and the genomic positions for a set of affymetrix probes found to be upregulated in a microarray experiment:

```
library(biomaRt)

human = useMart("ensembl", dataset="hsapiens_gene_ensembl")

upregulated = c('215984_s_at', '203174_s_at', '215984_s_at')

getBM(attributes=c("ensembl_gene_id", "chromosome_name", "start_posi-
tion", "end_position"), filter="affy_hg_u133_plus_2",
values=upregulated, mart=human)
```

An example of genomic visualisations using 'genomeGraphs' library retrieving gene annotations from ensembl mart:

```
library(GenomeGraphs)
Loading required package: grid
data("dummyData", package="GenomeGraphs")
minbase = min(probestart)
maxbase = max(probestart)

human = useMart("ensembl", dataset="hsapiens_gene_ensembl")
Checking attributes and filters ... ok

genesplus = new("GeneRegion", start = minbase, end = maxbase,
strand = "+", chromosome = "3", biomaRt=human)

genesmin = new("GeneRegion", start = minbase, end = maxbase,
strand = "-", chromosome = "3", biomaRt=human)
```

For more information see

<http://www.bioconductor.org/packages/1.8/bioc/html/biomaRt.html>.

7.2 Taverna

The Taverna project aims to provide a language and software tools to facilitate easy use of workflow and distributed compute technology within the eScience community. BioMart is available as Taverna data source.

For more information see <http://taverna.sourceforge.net/>.

7.3 Galaxy

Galaxy is an easy-to-use, open-source, scalable framework for tool and data integration. BioMart web interface (MartView) is fully functional within Galaxy for data retrieval and subsequent analysis.

For more information see <http://main.g2.bx.psu.edu/>

7.4 Ensembl

The Ensembl project generates and maintains genome annotation on a wide variety of eukaryotic genomes. BioMart software is fully integrated into the main Ensembl website as the BioMart interface and this integrated software and data is freely available for local installation.

For more information see <http://www.ensembl.org/>.

7.5 GMOD

The Generic Model Organism Database (GMOD) project aims to provide a free set of software for creating and administering a model organism database including genome visualization and annotation and literature curation. BioMart is part of this project. Currently the main method of interplay between GBrowse and BioMart is a tool developed for the GMOD project by Don Gilbert called *gff2biomart5.pl*. This tool takes well-formed GFF3 and creates MySQL BioMart tables for bulk uploading to the database.

For further information see <http://www.gmod.org/>.

7.6 Bioclipse

The Bioclipse project is aimed at creating a Java-based, open source, visual platform for chemo- and bioinformatics based on the Eclipse Rich Client Platform (RCP). Bioclipse, as any RCP application, is based on a plugin architecture that inherits basic functionality and visual interfaces from Eclipse, such as help system, software updates, preferences, cross-platform deployment etc. BioMart is available in Bioclipse as a data source.

7.7 WebLab

WebLab is a multifunctional bioinformatics analysis platform integrating diversified tools and data sources with unified, user-friendly web interface.

For further information see <http://weblab.cbi.pku.edu.cn/>.

END OF DOCUMENTATION.