BioMart 0.9.0 User Manual

May, 2014

Contents

1. Prerequisites

Software: Java 1.6, Ant and Git
OS: major Linux distributions
Server: min. one GB memory, three GB for better performance

2. Download BioMart

From the directory in which you wish to install BioMart, run the following command:

```
git clone https://github.com/biomart/BioMart.git --branch 0.9.0
```

3. Compile and Build BioMart

From the directory in which you installed BioMart, run the following command:

```
ant
```

This will compile BioMart source and build the distribution.

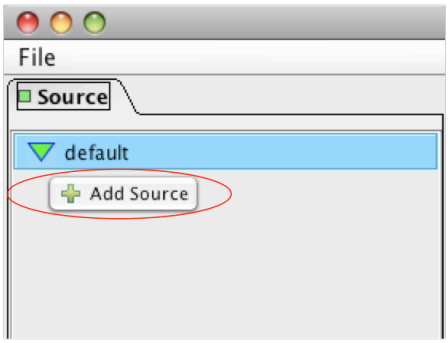
4. Quick Start

This section will show you how to add the *MSD protein structures* dataset and deploy the BioMart server.

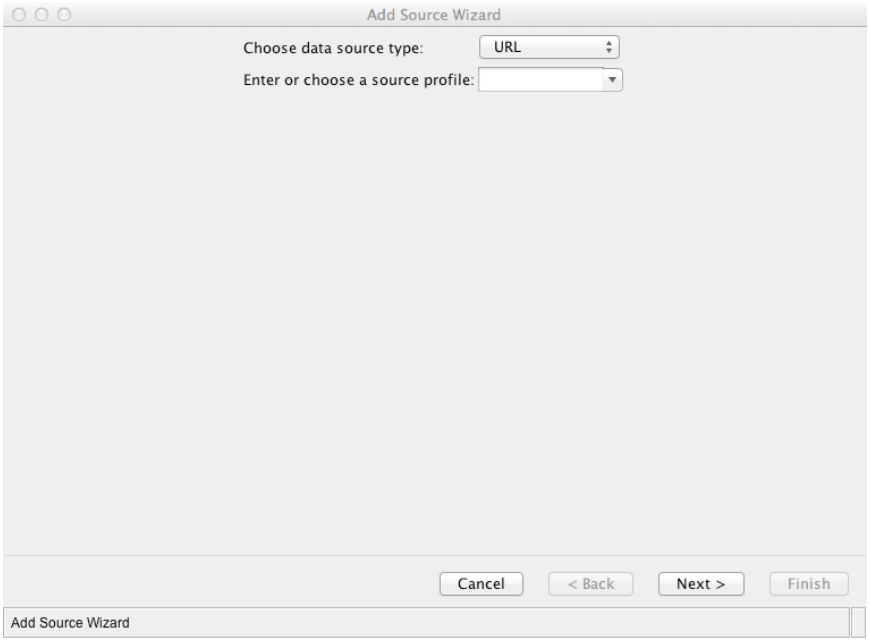
Run MartConfigurator with the following command in the directory of your installation:

```
./dist/scripts/martconfigurator.sh
```

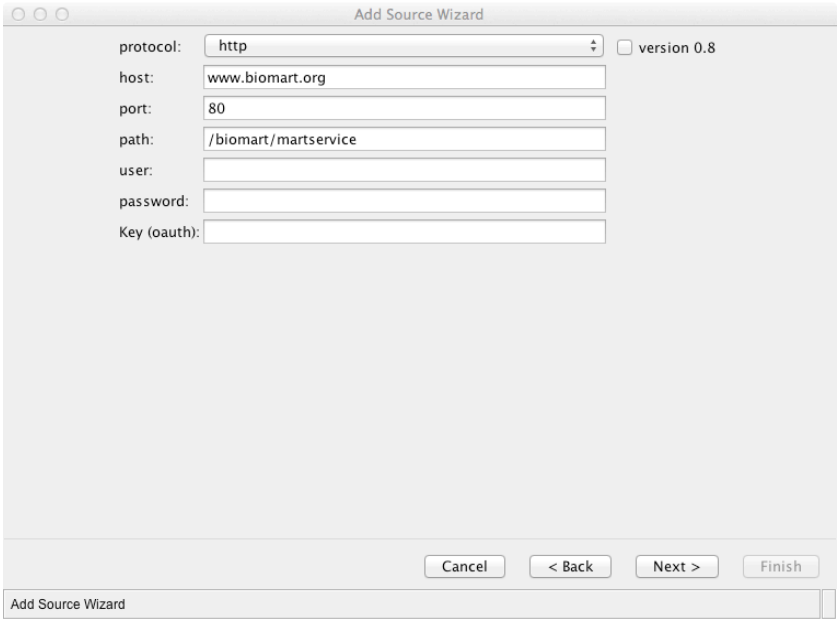
In the MartConfigurator window, click *Add a new data source* in the upper right corner:



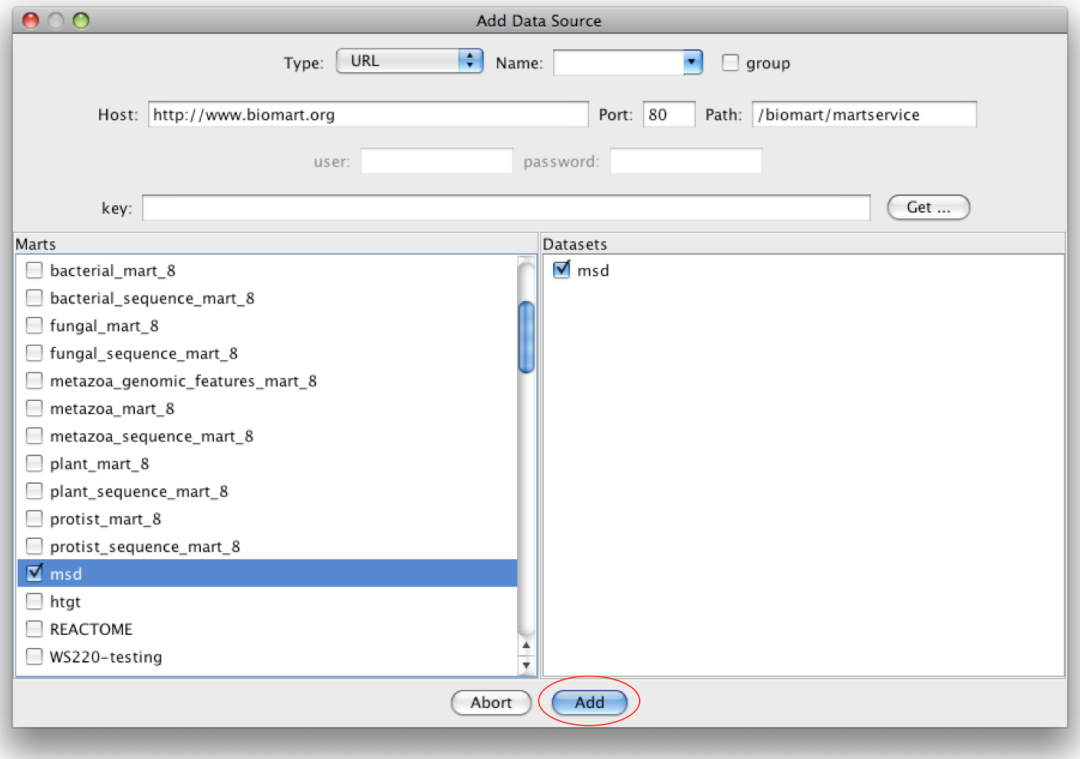
In the “Add Source Wizard” window that appears, click the *Next…* button



and *Next* button again to to connect to BioMart central portal:

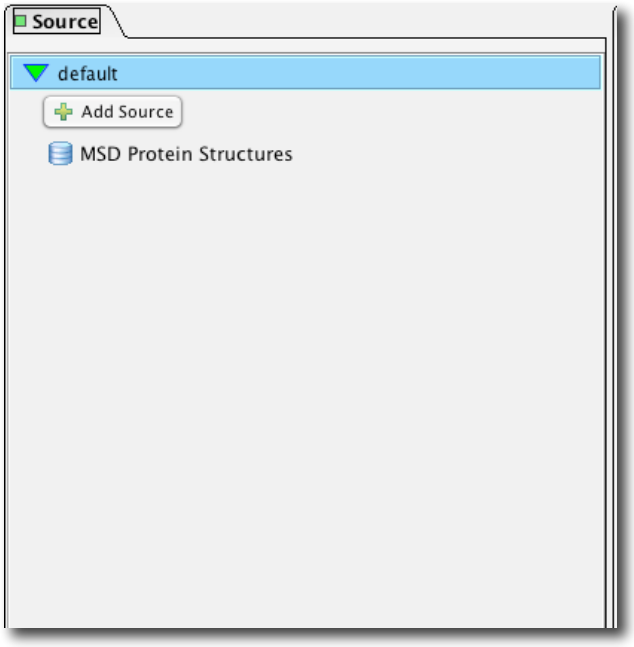


A list of available marts will appear in the left panel. Click on *msd* to select and check it:

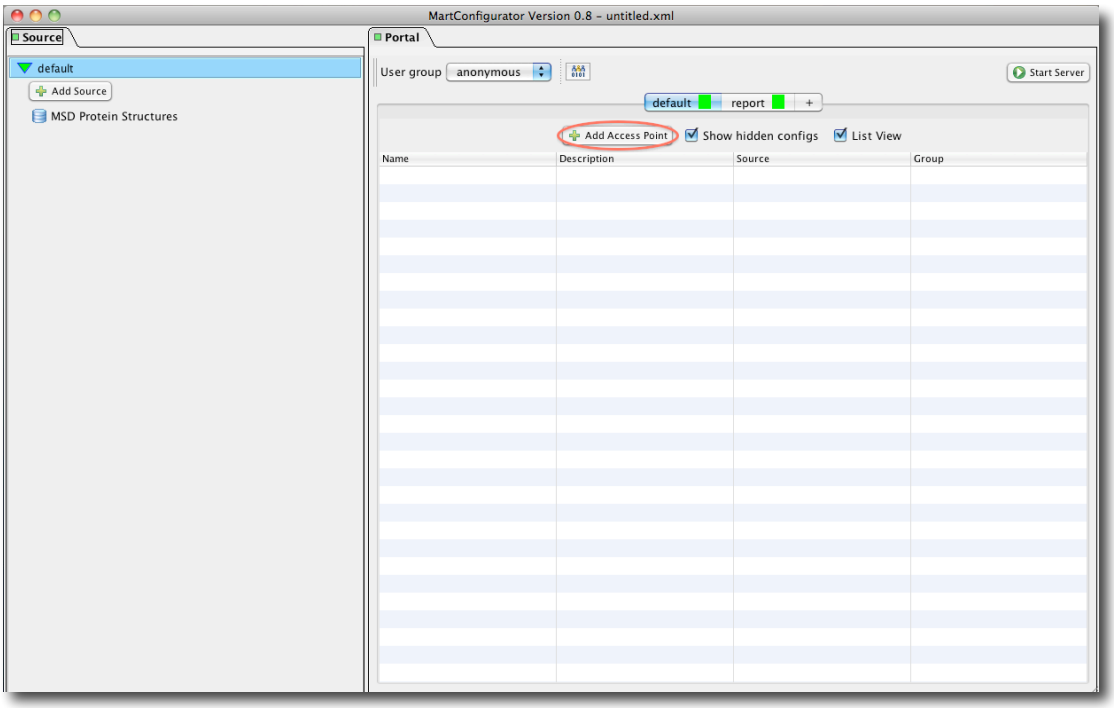


The *msd* dataset will appear in the right-hand panel and will be checked. Click the *Add* button at the bottom of the window to add the data sources.

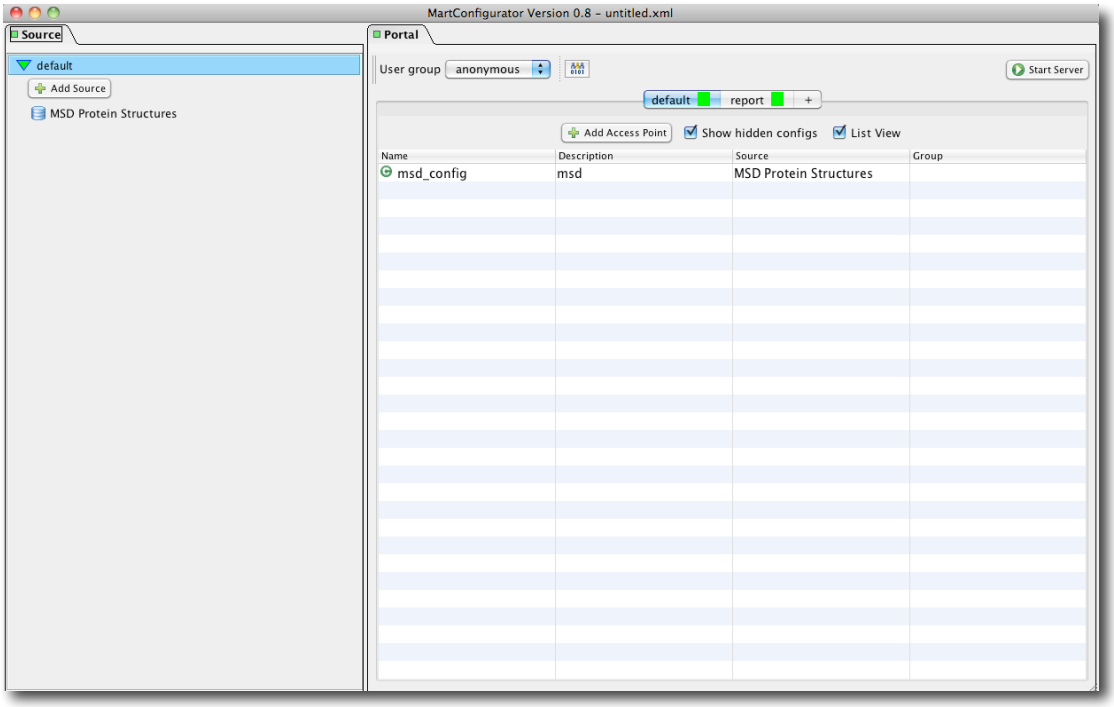
The new data sources will appear in the left-hand panel on the MartConfigurator window.



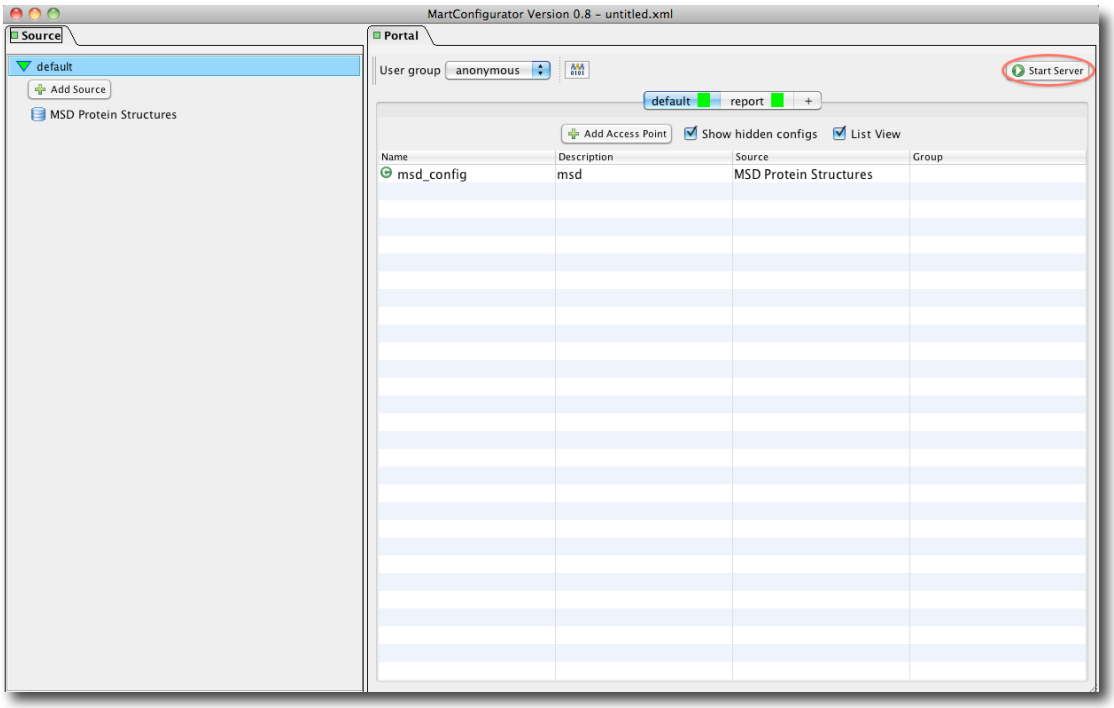
In the right-hand panel, click the *Add Access Point* button:



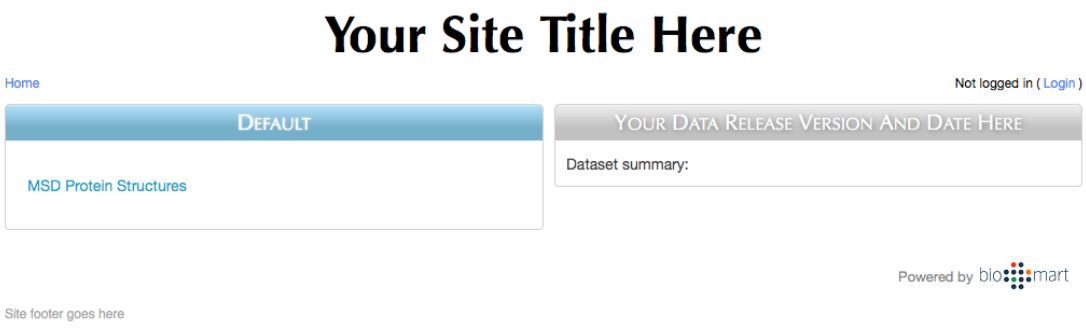
You will be prompted to choose a main data source. Click on *MSD protein structures* and click *OK*. When prompted to choose a name, click *OK* to use the default. A new access point will be created and shown in the right-hand panel:



To test the registry you can now click on the *Start Server* button in the upper-right:



You will be prompted to save the registry file you've created, and after doing so the server will be deployed on your local machine, port 9000. Your web browser should open to the home page automatically when the local server is ready:



You can also deploy BioMart server from the command line on a server. First, from the *File* menu, save your registry as *quickstart.xml*. It is recommended that you save them to the *registry* subdirectory of the directory where you installed BioMart.

Next, you must configure the BioMart server. In a text editor open the *biomart.properties* file, located in the *dist* subdirectory of the directory where you installed Biomart.

In the "HTTP settings" section, change the *http.host* property to 0.0.0.0 and change the *http.port* property to the port on which you would like the non-

secure web server to run (default 9000). Remove "*#*" at the beginning of *http.url* line, and set it to the public URL:port from where your BioMart server is accessible, for example:

```
http.url = http://your.domain.org:9000
```

In the "BioMart settings" section, change the *biomart.registry.file* property to point to "quickstart.xml. Change the *biomart.registry.key.file* property to point to ".quickstart". **Note:** Make sure that the full path to these files is correct based on where you saved them. By default, this will be in the *registry* subdirectory of the directory where you installed BioMart.

To deploy BioMart, from the directory of your installation run the following command:

```
./dist/scripts/biomart-server.sh start
```

To stop the server, use the command:

```
./dist/scripts/biomart-server.sh stop
```

It may take several minutes before the server starts and the site is viewable.

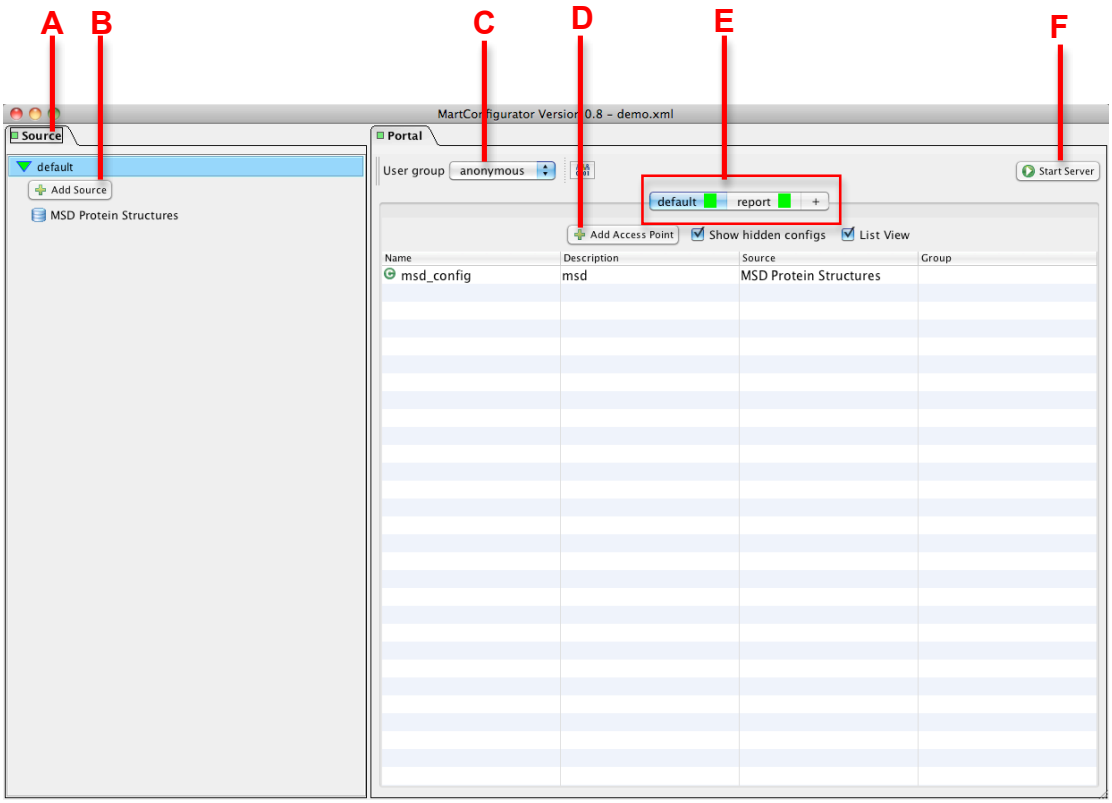
Once the server is started, please navigate your browser to the proper host and port you just set up, e.g.

```
http://your.domain.org:9000
```

5. Using MartConfigurator

5.1 The MartConfigurator window

On first starting up the MartConfigurator window will look like this:



The *File menu* (either at the top of the window or the top of the screen, depending on your system) allows you to create a new registry, open an existing registry, or save the registry that is currently being worked on. Saving a registry for the first time automatically encrypts any passwords and generates a key file needed to decrypt it. The key file name is given based on this convention: it starts with a period (“.”), followed by the registry file name with “.xml” being removed. For example, key file name will be *.myregistry* when the name of the registry file is *myregistry.xml.* Using the *Save as* option to rename a file will also generate a new key file. The key file with a proper key is needed to decrypt all passwords the next time the registry is opened with MartConfigurator or is used to deploy a BioMart server.

The *Source* panel (A) contains a list of dataset sources that are in this registry. You can add a new data source by clicking on *Add Source* (B).

The *Portal* panel contains information about the configurations (configs) in the registry. The controls in the Portal tab will be inactive until a data source

is added to the registry.

The visibility of each config can be set differently for different user groups. The active user group whose visibility settings are displayed and configurable is shown in drop down (C).

Access points are organized into GUItabs, which determine how they are presented to the user. The current GUItabs are listed in (E), and a new GUItab can be added by pressing the “+” sign at the right side of the tab bar. Initially there are two GUItabs, *default* and *report*. The *report* GUItab allows creation of report pages.

Within a GUItab configs can be added by pressing the button labeled *Add Access Point* (D).

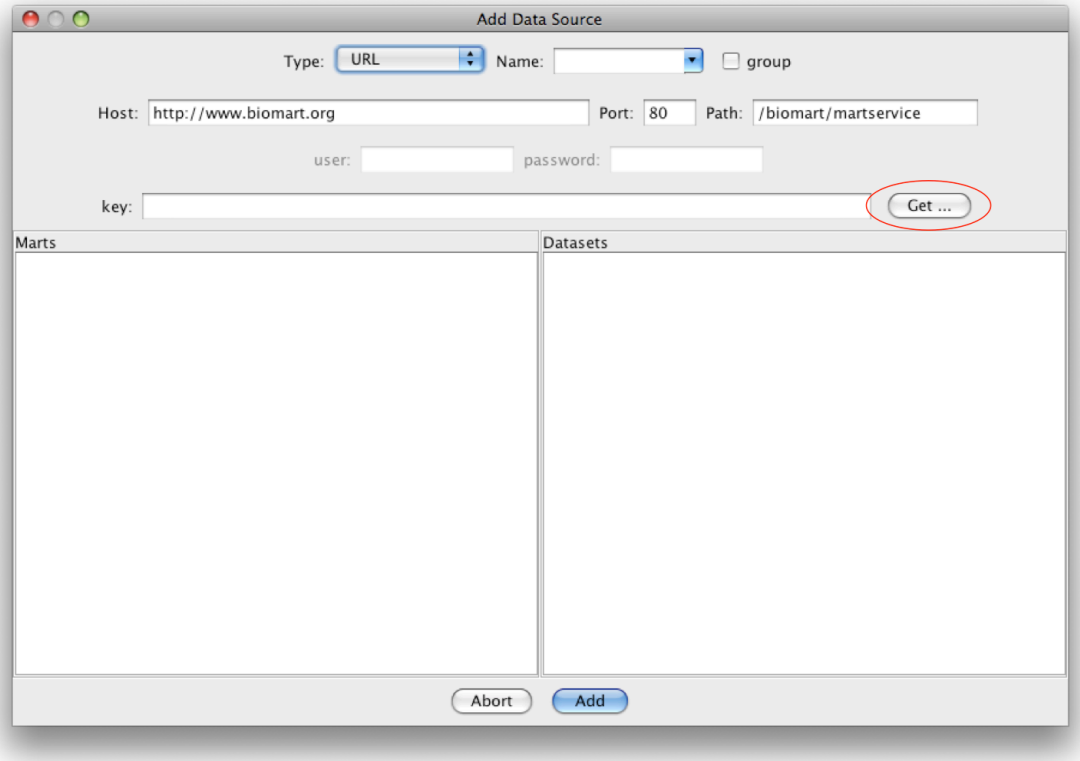
The current registry can be deployed locally for testing using the *Start Server* button (F).

5.2 Importing data sources

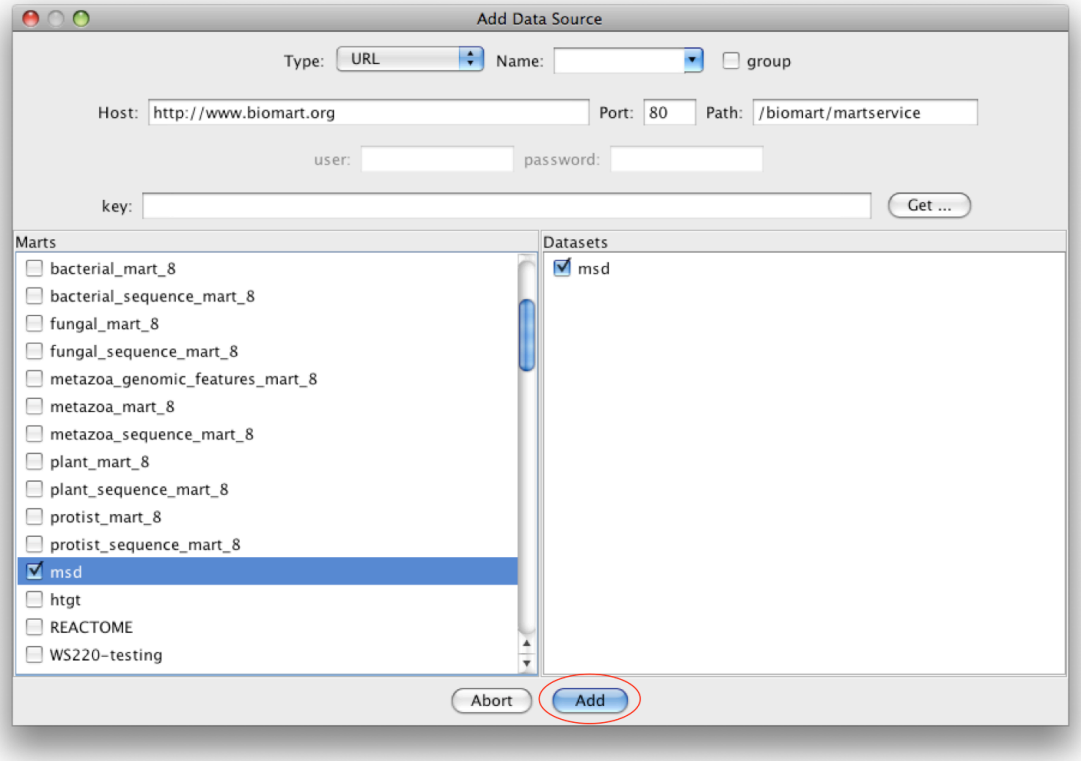
There are three types of data sources that can be imported into MartConfigurator: URL, RDBMS [Relational Mart or Source Schema (aka, 'Virtual Mart')], and data sources configured in a Registry File. For all of them, start by clicking the *Add source* button (C).

5.3 Adding a new URL Mart

By default the import type is set to URL. To import from URL, enter your connection parameters and click the *Get…* button in the upper right. A list of marts available on the server should then appear in the left panel.



Selecting a Mart in the left panel will show a list of the datasets in this Mart. By default all datasets will be checked. Clicking on an individual dataset will toggle whether or not it is checked.



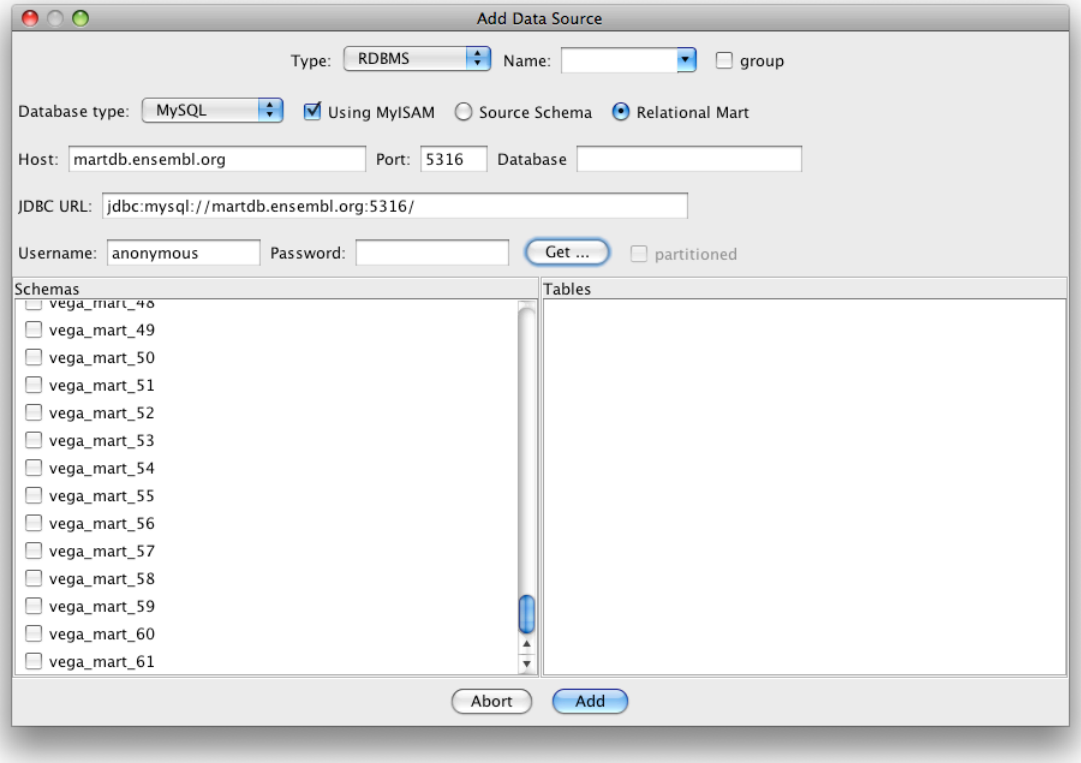
Clicking the *Add* button at the bottom of the window will import each checked dataset as a separate data source.

5.4 Adding a new RDBMS Source

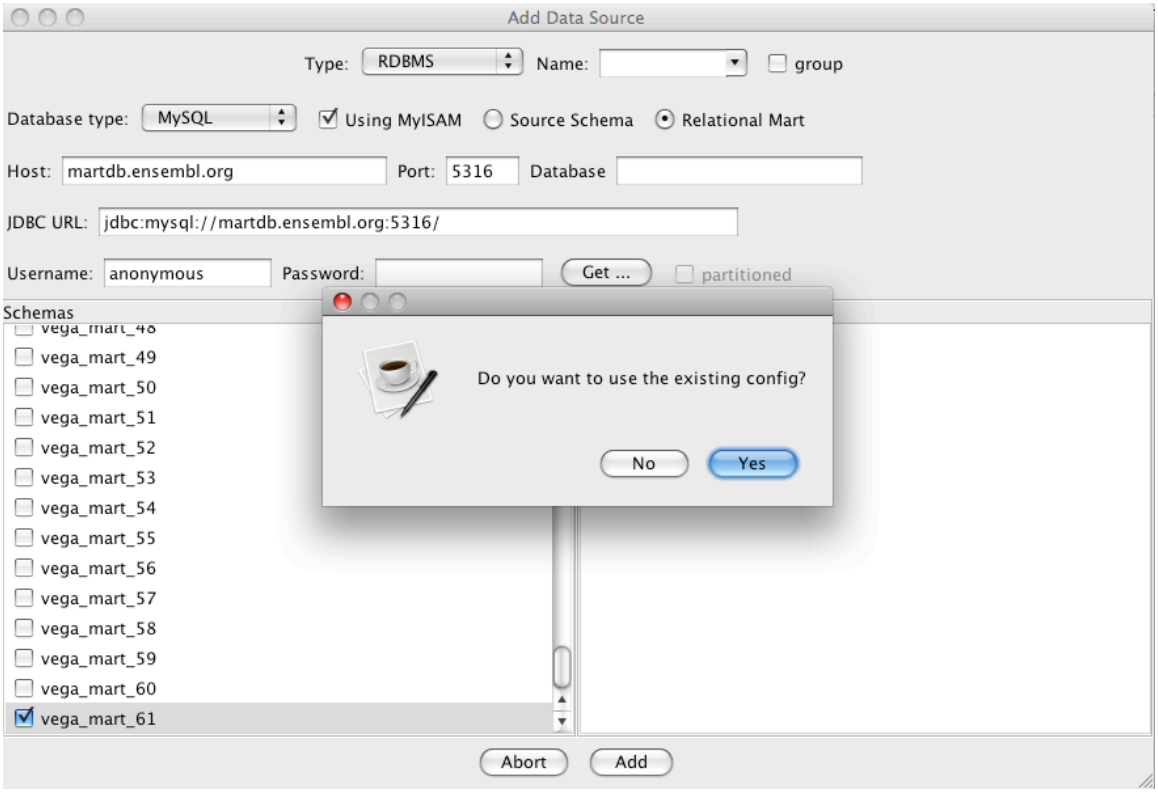
The RDBMS type sources are divided into two sub-types, *Relational Mart* and *Source Schema.*

5.4.1. Adding a new Relational Mart

A *Relational Mart* can be used to import an already existing materialized mart. To add a new Relational Mart, first add a new data source by clicking on the *Add Source* button. Select *RDBMS* from the *Type* dropdown, select *Relation Mart* radio button and then select your database type from the *Database type* dropdown. Enter your database connection parameters, including username and password, and then click the *Get…* button. A list of available marts will show in the left panel.



Once you have made your selection you will be prompted to use the existing config:

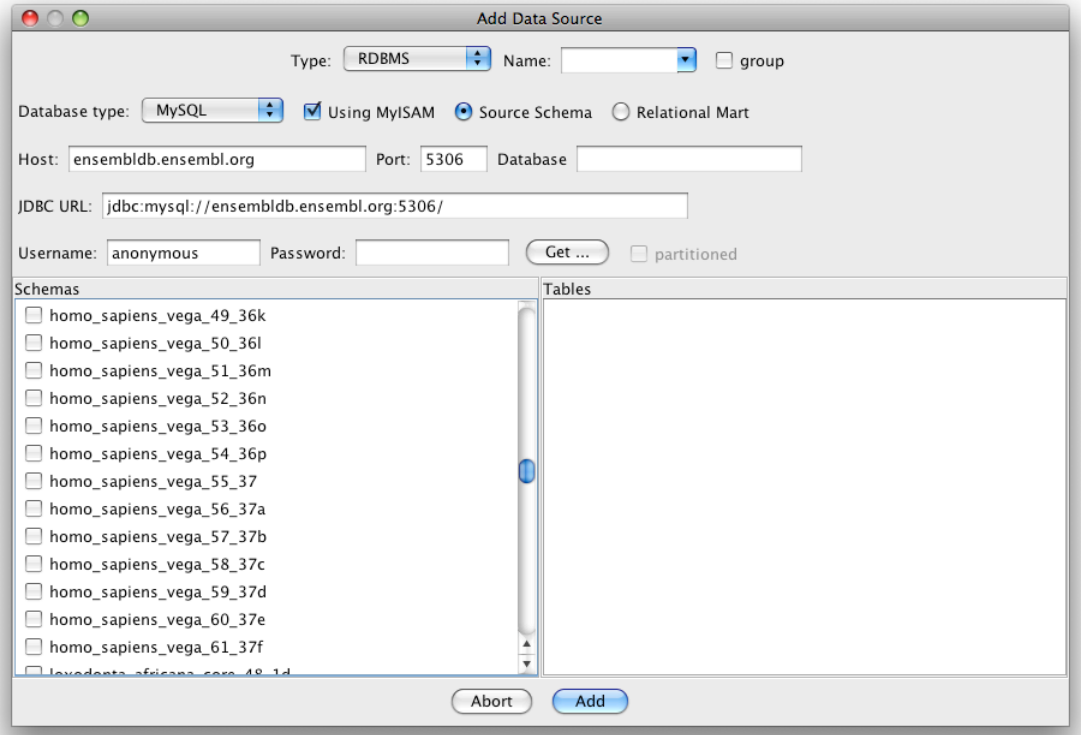


Select *Yes* and then click the *Add* button without changing any other settings to import the configuration from existing mart.

To add a new data source from a materialized mart and ignore the existing configuration, follow the instructions for importing a *RDBMS-Relational Mart*, but select *No* when asked if you want to use the existing config. Then select a top-level main table (eg, gene table in Ensembl mart) from which to create your new data source and press *Add*. Note that using this method you can only select one dataset at a time.

5.4.2. Adding a new Source Schema ("Virtual Mart")

Source Schema option in RDBMS type is for importing relational sources based on the 3NF schema by dynamically creating a non-materialized ("virtual") mart. To add a new virtual mart based on a source schema, first add a new data source by clicking on the *Add Source* button. Select *RDBMS* from the *Type* dropdown, select *Source Schema* radio button and then select your database type from the *Database type* dropdown. Then fill in the *Host*, *Port, Database, Username,* and *Password* parameters to connect to your database server (the *Database* field is optional for MySQL servers). The *JDBC URL* field is populated automatically and should not be modified.



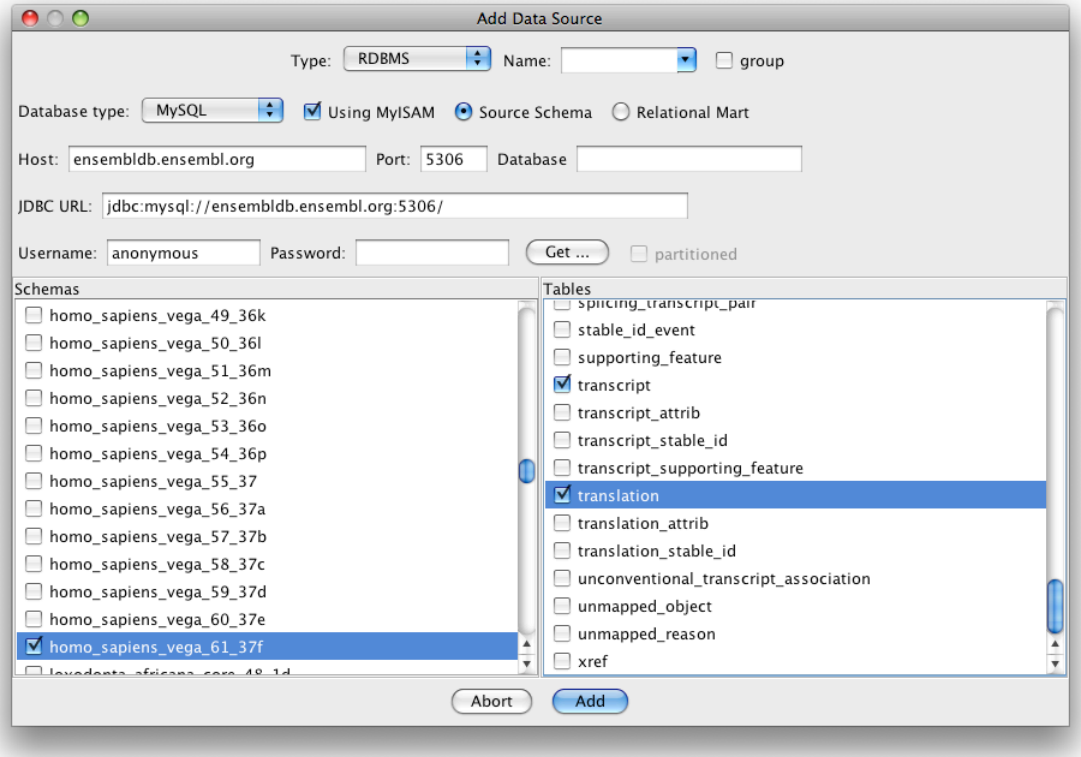
Clicking the *Get…* button will connect to your database server and show a list of available schemas in the left panel. Select a schema to show a list of available tables. Note that if you did not specify a schema when creating your database, your tables will be in the default schema for your platform:

MySQL: does not have schema, will be the same as the database name

PostGreSQL: “public”

Oracle and DB2: the username of the user who created the database

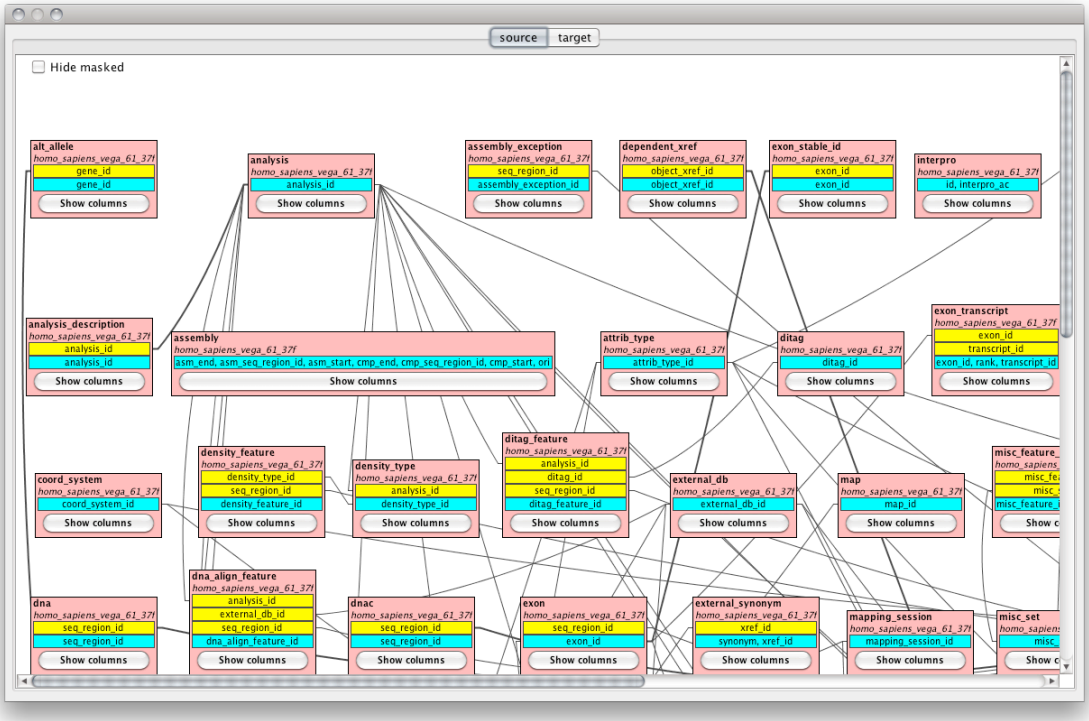
SQL Server: “dbo”



In the right panel select the table(s) that will become you main table(s) and click *Add* to create your data source.

5.5 Manipulating a Mart using the Schema Editor

Various properties of a virtual mart can be customized by right-clicking on the source and choosing *Schema Editor*. This will bring up the Schema Editor window.



The Schema Editor shows database tables (pink boxes) as well as the relations between these tables (connecting lines). The columns in a given table can be shown by clicking the *Show columns* button for that table.

A table, column, or relation can be hidden by right-clicking on the object and selecting *hide*. This will effectively remove that table, column, or relation from the BioMart representation of the database. It can be undone by right-clicking the object again and selecting *unhide*. Relations can also be changed between 1:1 and 1:M, which will have an effect on how source tables are materialized.

The Schema Editor window for virtual marts comprises two tabs, *source* and *target*. By default, *source* is selected. This shows the tables as they exist in the underlying source database. The *target* tab shows the tables that will be created by materializing. Changes made in the *source* view, such as hiding tables and columns or changing relation type, will be reflected immediately in the *target* view.

In the target view, right-clicking on a table and selecting *Explain Table* will show the series of joins that is performed to transform the source schema to the materialized schema.

5.6 Materializing a Virtual Mart

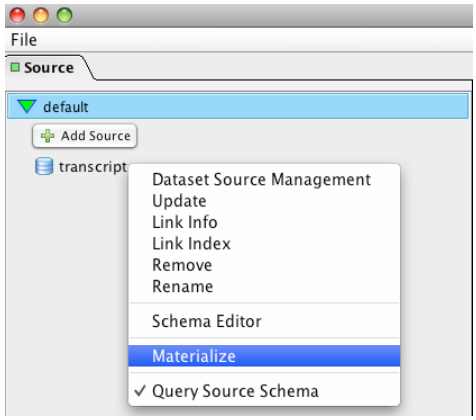
Materializing a virtual mart considerably improves querying speed for large databases. To do this you must first add a virtual mart to your data sources, as described in the previous section.

First, you must have MartRunner running. From the directory in which you installed BioMart, run the following command:

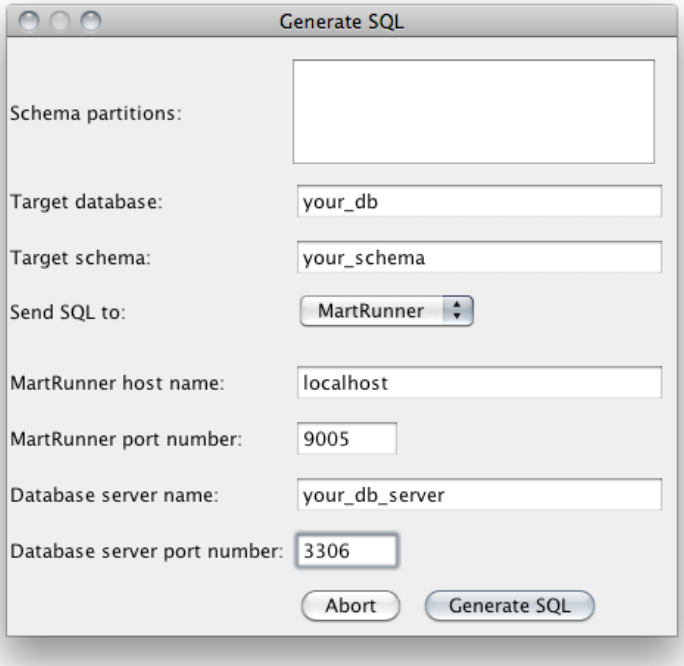
```
./dist/scripts/martrunner.sh 9005
```

Where “9005” is a port that is free on your machine.

Next, in the MartConfigurator window, right-click on the virtual mart and select *Materialize* from the drop-down menu



The “Generate SQL” window will appear, with all of the text fields blank. These must be filled in with the correct connection parameters.



The proper information for the *Target database* and *Target schema* fields differ depending on the database server type:

MySQL– Target database and target schema must be the same, and different than the original source database and schema. The database must exist on the server.

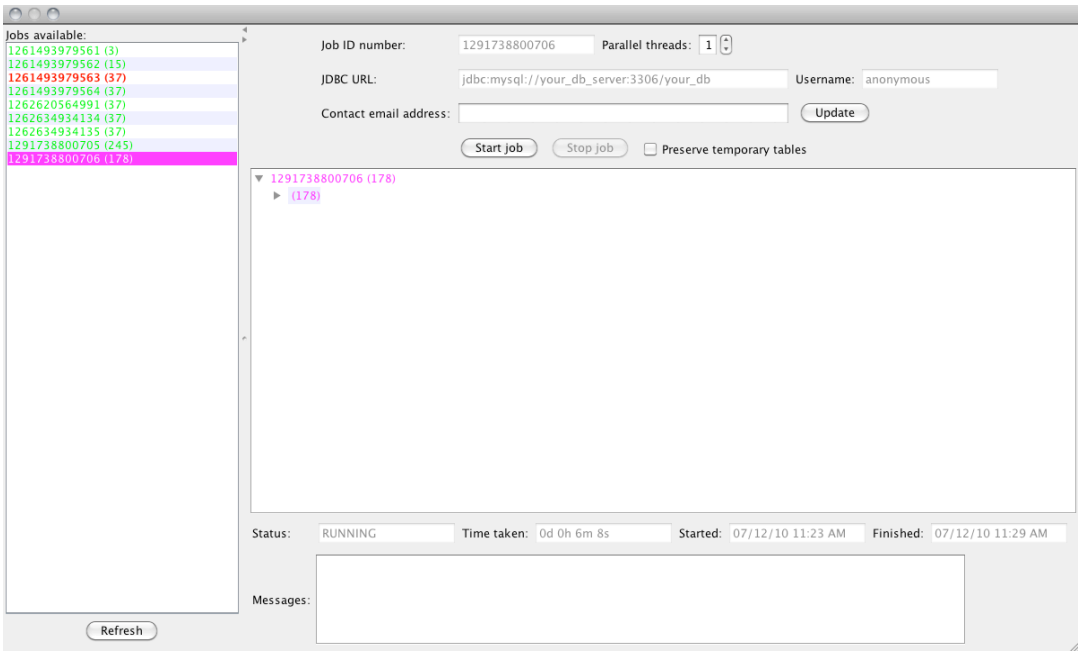
SQL Server – Target database and target schema both must exist on the server. The original source schema should not be used.

PostgreSQL, Oracle, and DB2 – Target database must be the same as the original source. The target schema should exist within this database, and should be different than the original source schema.

The *MartRunner host name* is localhost and the *MartRunner* port number is whatever you chose when executing the martrunner command (e.g 9005).

The *Database server name* and *Database server port number* should be the same as the database server for your non-materialized data source.

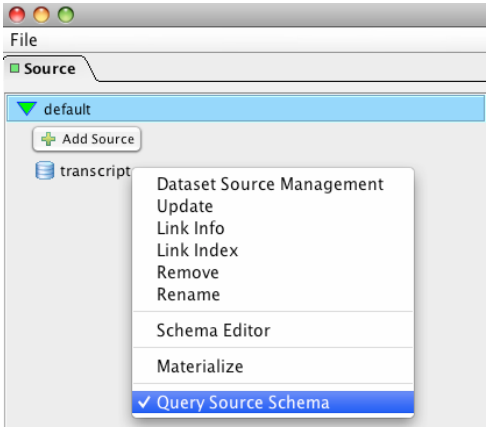
When these fields have been entered, click *Generate SQL*. This will bring up the MartRunner jobs window.



In the left-hand panel, labeled *Jobs available*, there will be a list of numbers. Green numbers are successfully completed jobs, red numbers are jobs that aborted due to errors, pink numbers are jobs that have not been started, and blue numbers are jobs that are in progress.

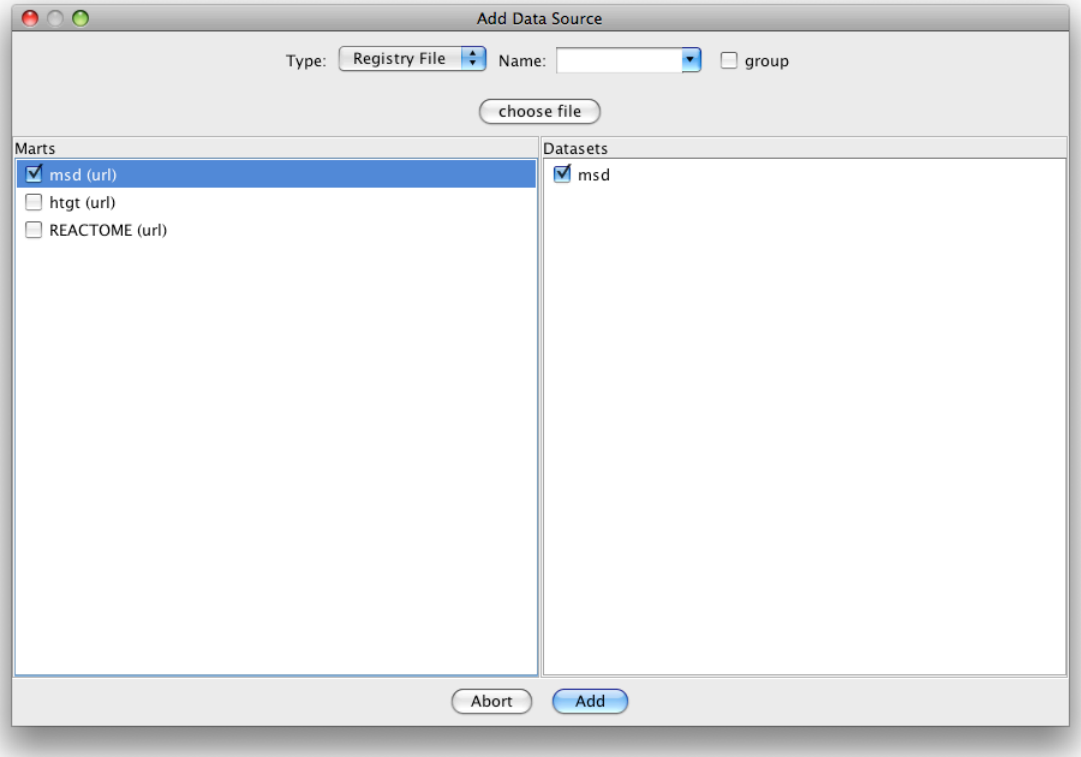
Your job should be the last on the list and in pink (if this is your first time materializing a schema, it will be the only job). Select it by clicking on the entry in the left-hand panel, then click *Start job*. Depending on the size of your database, this may take several hours. You can update the status of the job by clicking on the *Refresh* button in the lower left corner of the window. When the job number turns green it is complete, and you may close the window.

Once the job is complete, you can set the database to query the materialized mart by right-clicking the data source and unchecking the *Query Source Schema* option.



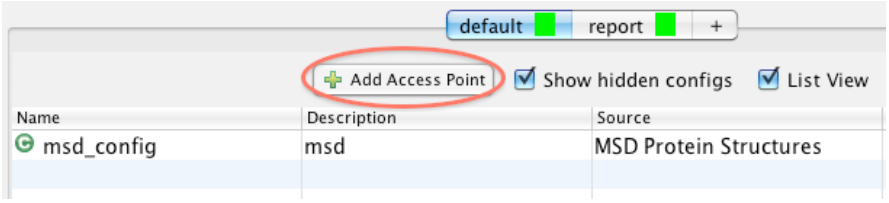
5.7 Adding marts from Registry File

The *Registry File* option in Add Source panel facilitates the import of a BioMart v0.7 registry file that in turn points to various *MartDBLocation* and *MartURLLocation* based sources. The xml file can be specified using the *choose file* button and subsequently selecting all or a selection of marts to be imported into MartConfigurator using *Add* button. The software will automatically run the backwards compatibility to transform BioMart v0.7 sources to v0.8.



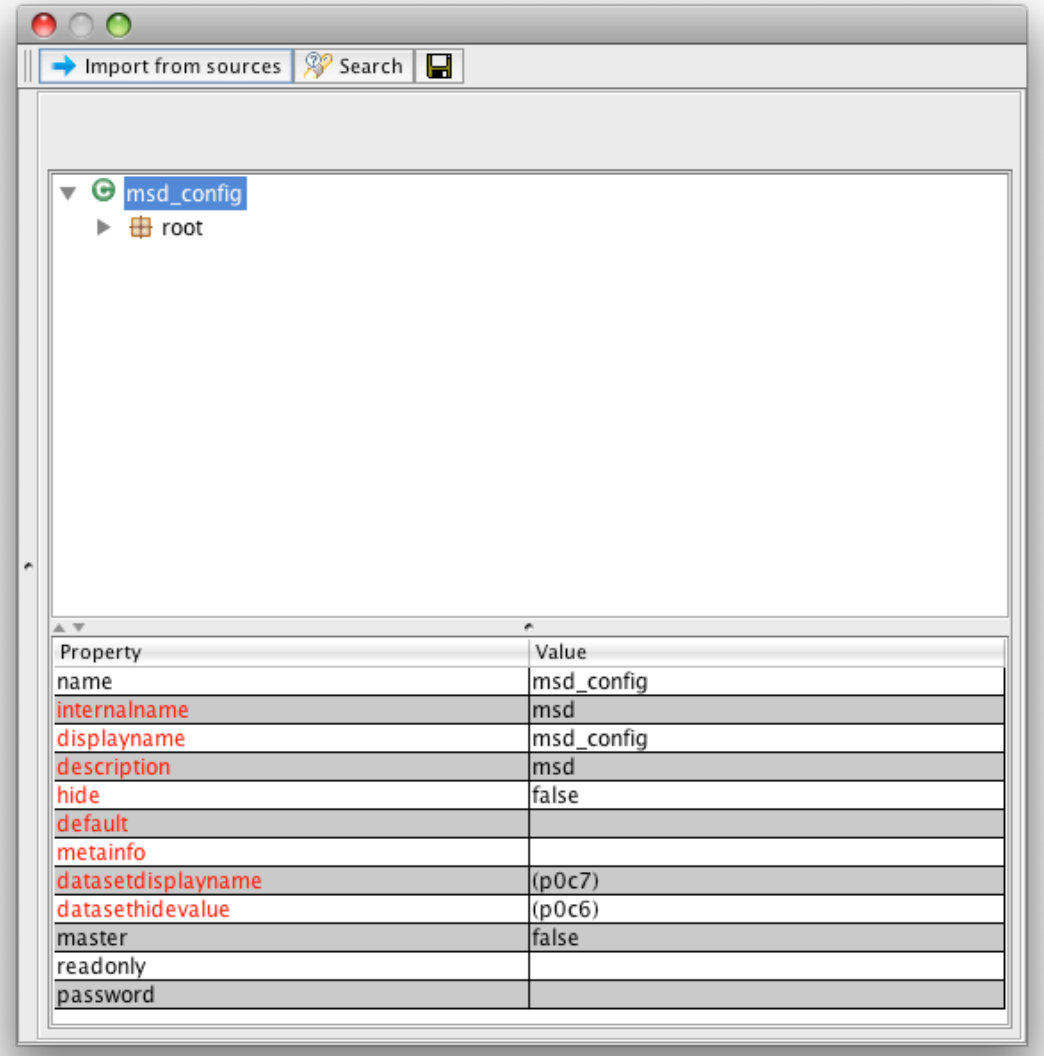
5.8 Creating and modifying an access point

To create an access point click on the *Add Access Point* button in the *Portal* section:



You will be given a list of the existing data sources to choose which one you would like to make an access point for. After giving the new access point a name of your choice, it will appear in the GUI tab.

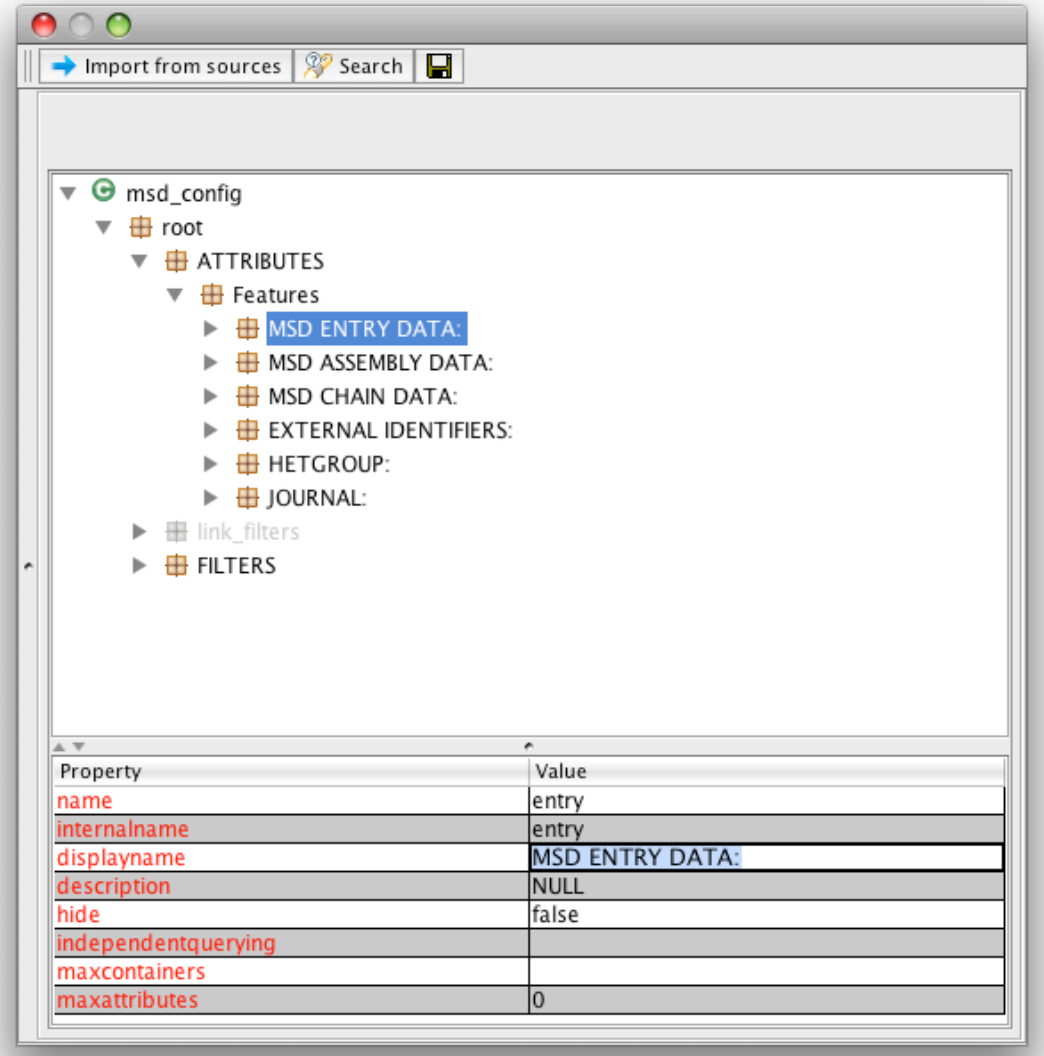
Double-clicking on the access point icon will open a new window that allows you to modify the access point.



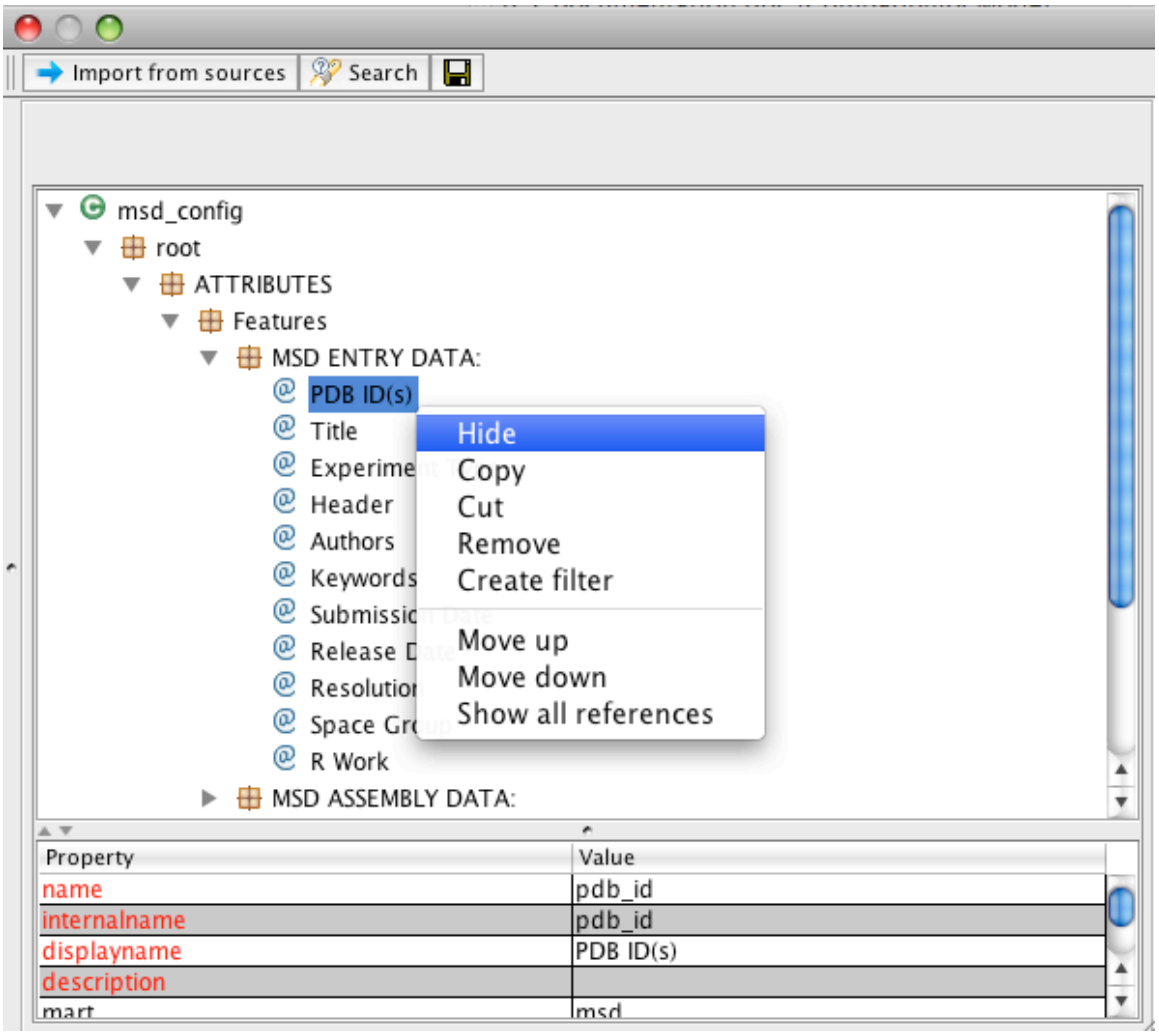
The top half of the window shows a tree view of the objects in the config. Containers can be expanded or collapsed by clicking on the triangle next to its name.

The bottom half of the window shows various properties of the highlighted object and their values.

The display name of any object (a container, attribute, or filter) can be changed by selecting that object (by clicking on it) and then double clicking the *displayname* property in the lower right-hand pane.

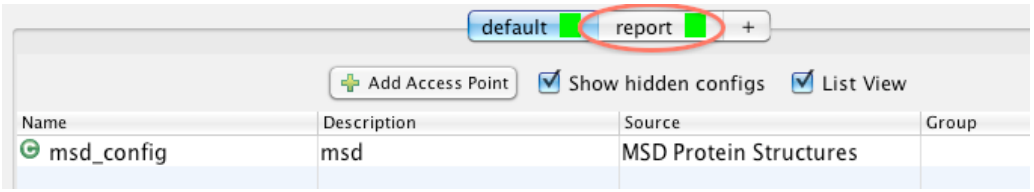


Any object can be hidden from the end user by right clicking on that object and selecting *Hide* from the menu.

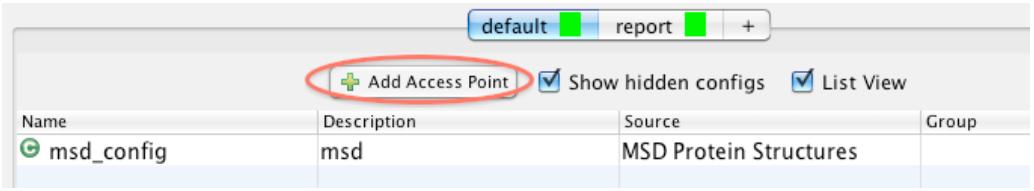


5.9 Creating a report

A report is a specialized type of access point that gives information based on a single attribute (key) that is used to retrieve information from our main source and other linked sources. Note that a report can only be created based on an existing config. To create a report, click on the *report* GUItab:

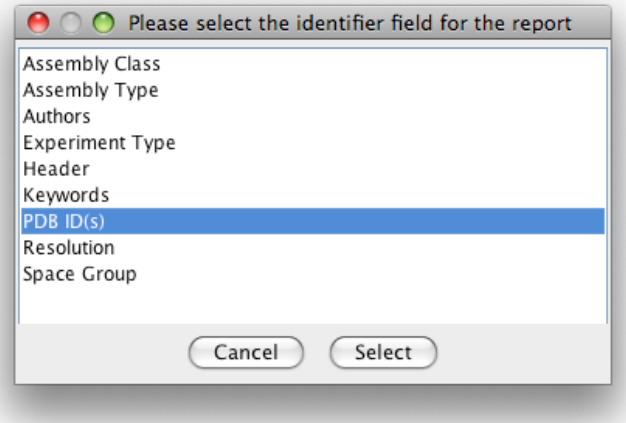


Then click on the *Add Access Point* button:



You will be prompted to choose a main data source. You must choose a data source that has at least one existing config; if you choose a data source with no config, an error message will appear.

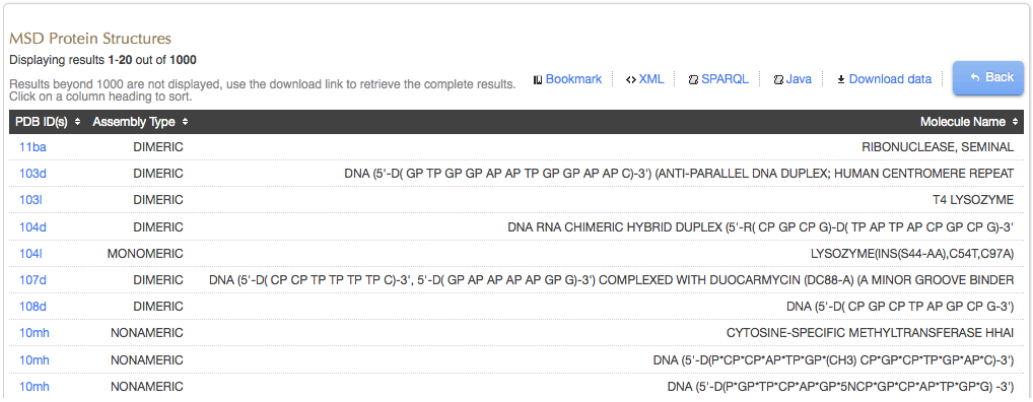
After being prompted to name the new config, you will be asked to choose an attribute to serve as the main attribute for the report page:



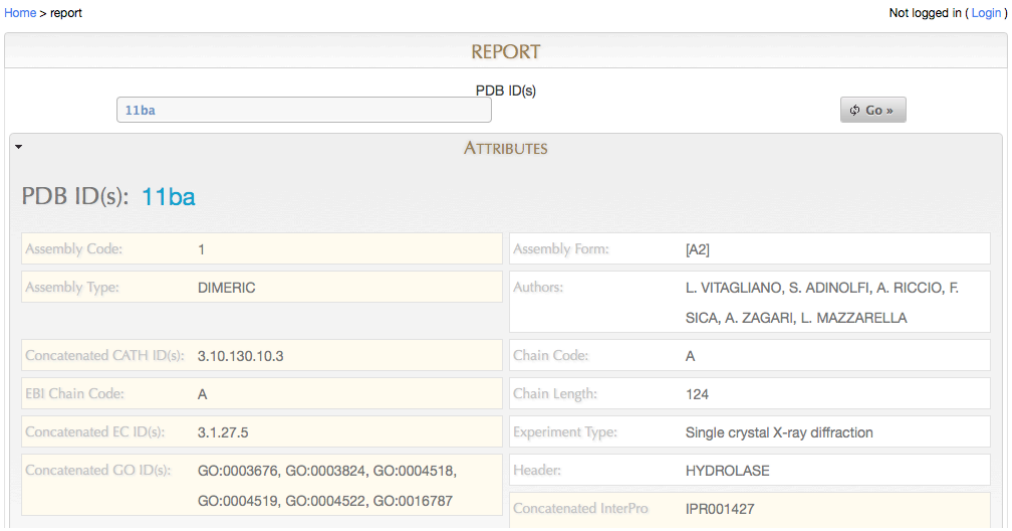
The attribute you select will be used as the “key” for the report page, and a hyperlink will be created on this attribute for linking to the report page. Clicking

the *Select* button will create the new access point.

Now, the report page would appear as hyperlink on the attribute selected (e.g PDB ID) in the Web GUIs. See example below:



Click on the PDB ID hyper link will bring up the report page as below:

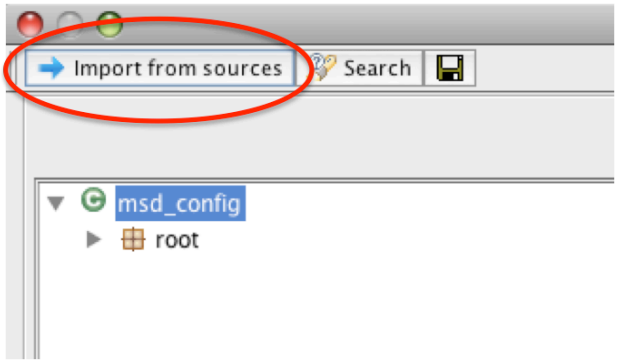


5.10 Creating links between sources

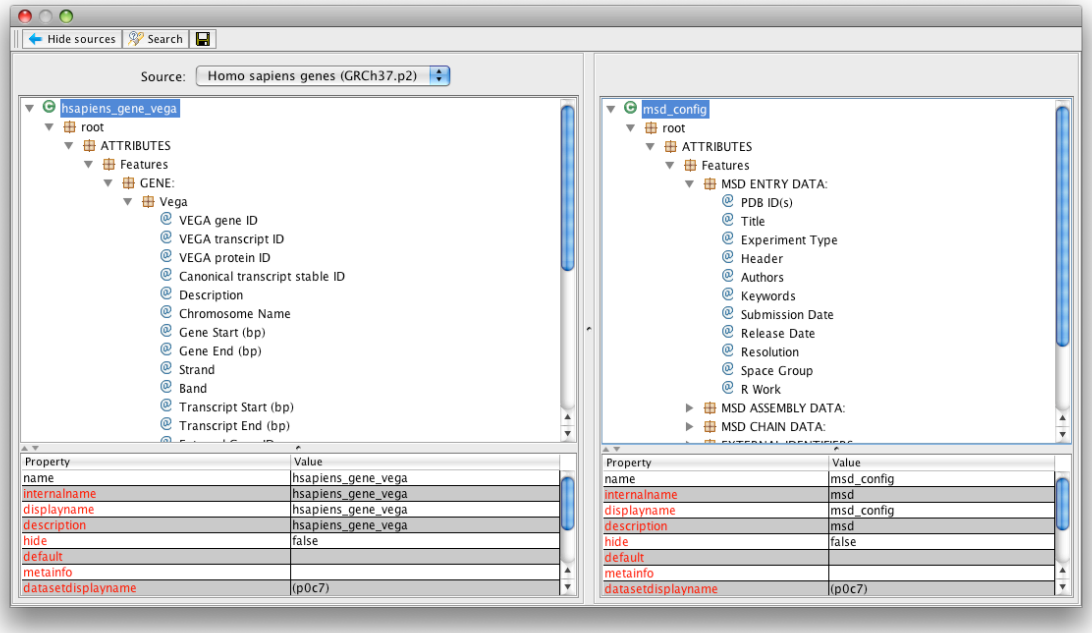
If two data sources contain common information (e.g. a Gene/Protein ID), this can be used to create a link, allowing filters and attributes from one data source to appear in the other. These are called “pointer attributes” and “pointer filters,” and the attribute or filter to which they point is called the “target.”

To add a pointer to an access point, double click on that access point in the portal tab to edit it.

In the top left corner of the editing window, click on the *Import from sources* button.

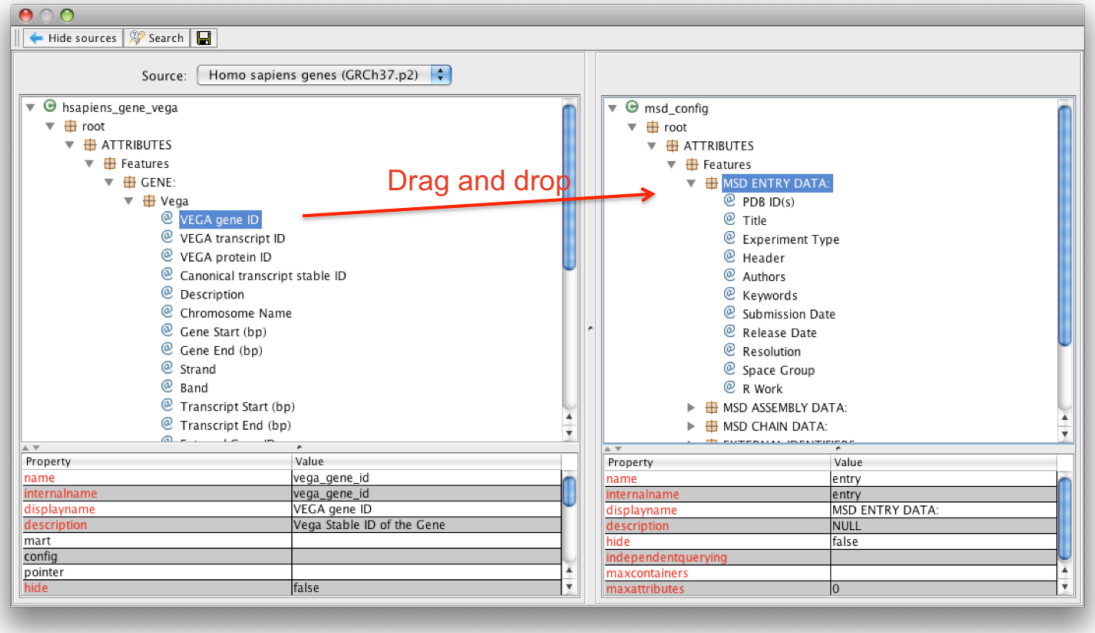


The window will divide into two similar halves. The right side represents the access point you are editing, and left side lists all of the sources.

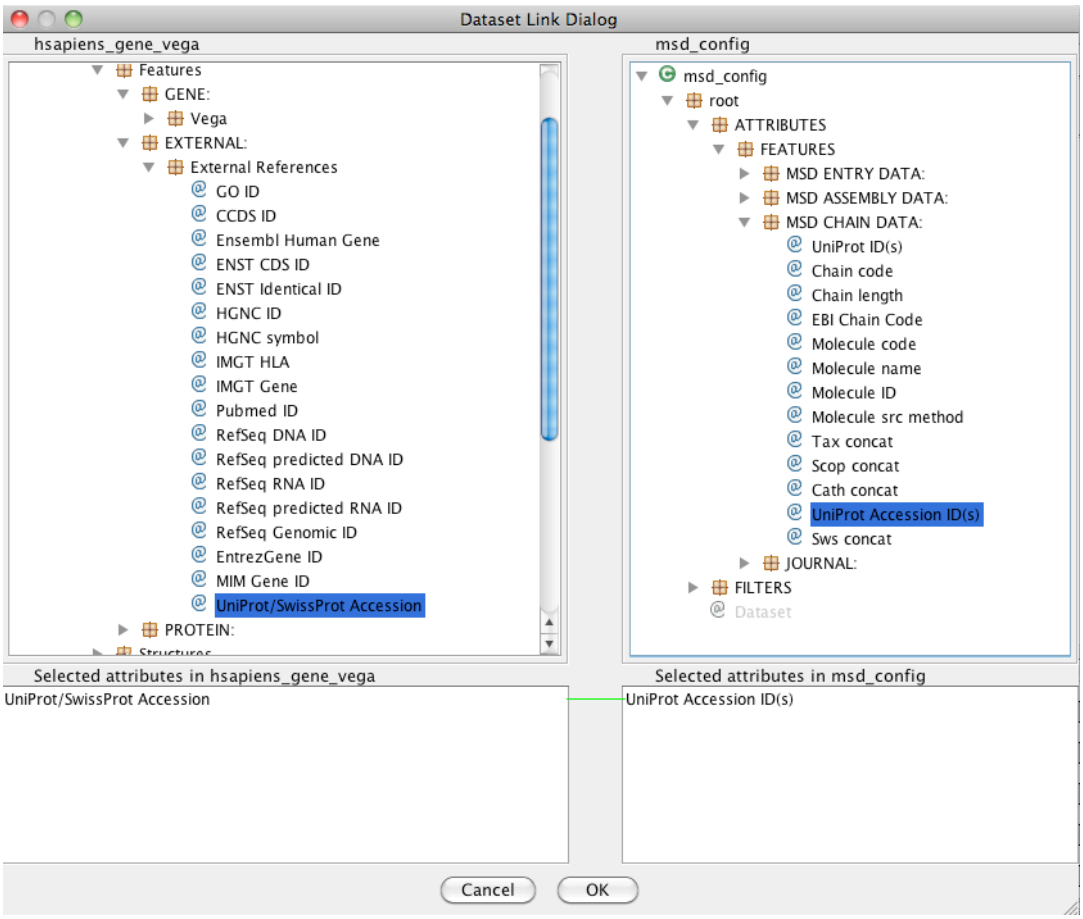


You can change the source using the drop-down menu at the top of left panel.

Once you have selected the desired source in the left panel, find the target attribute or filter for the pointer, and drag it to the container in the right panel where you want the pointer to be created.



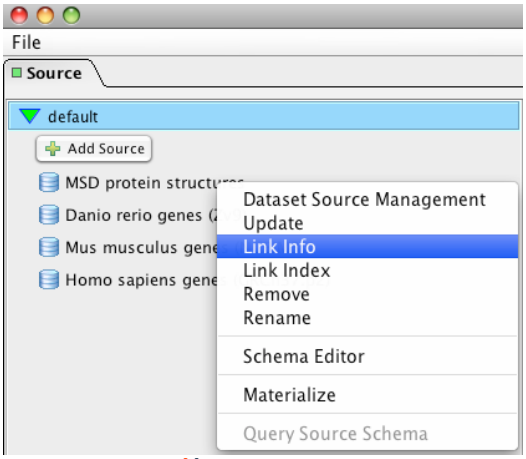
If no link exists between the sources, a Dataset Link Dialog will appear allowing you to create a link.



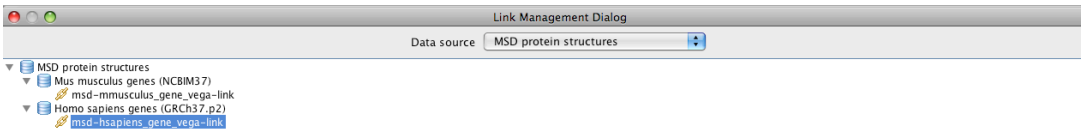
Select the attribute(s) on which to base the link in both sides by double clicking. Multiple attributes can be selected to form the link, but they must have the same number on each side. Lines are shown between to two lists of attributes to show which are being matched.

Click OK to create the link.

To see all the links for a source, right-click on the name of the source in the *Source* panel and select *Link Info:*



The *Link Management Dialog* window will appear, where you can see to which sources the selected source links. Clicking on a link name will then show information about this link in the lower panels:



MSD protein structures => Homo sapiens genes (GRCh37.p2)

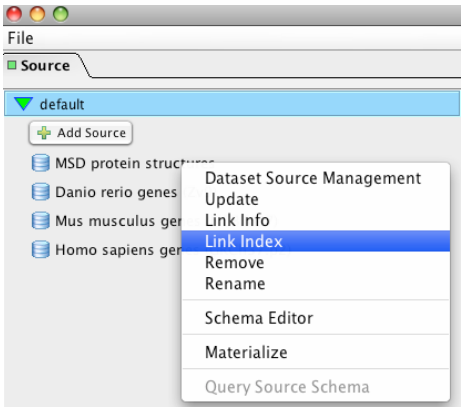
Attribute	Value
name	msd-hsapiens_gene_vega-link
internalname	msd-hsapiens_gene_vega-link
displayname	msd-hsapiens_gene_vega-link
description	msd-hsapiens_gene_vega-link
pointedmart	hsapiens_gene_vega
pointedconfig	hsapiens_gene_vega
attributes	UniProt Accession ID(s)
datasets	hsapiens_gene_vega

Homo sapiens genes (GRCh37.p2) => MSD protein structures

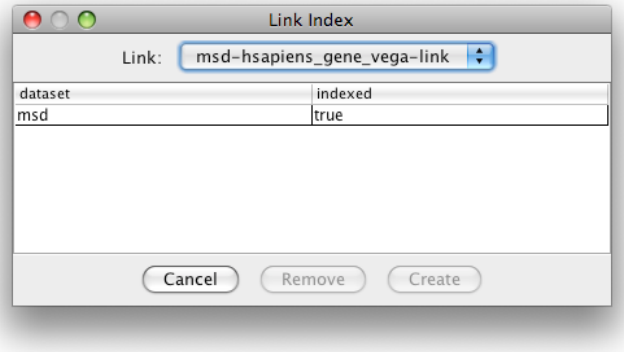
Attribute	Value
name	hsapiens_gene_vega-msd-link
internalname	hsapiens_gene_vega-msd-link
displayname	hsapiens_gene_vega-msd-link
description	hsapiens_gene_vega-msd-link
pointedmart	msd
pointedconfig	msd
attributes	UniProt/Swissprot Accession(s) [e.g. Q6VEP3]
datasets	msd

5.11 Creating a link index

Indices can be created for links in order to speed up searching. To do so, right-click on a data source in the *Source* panel and select the *Link Index* option:



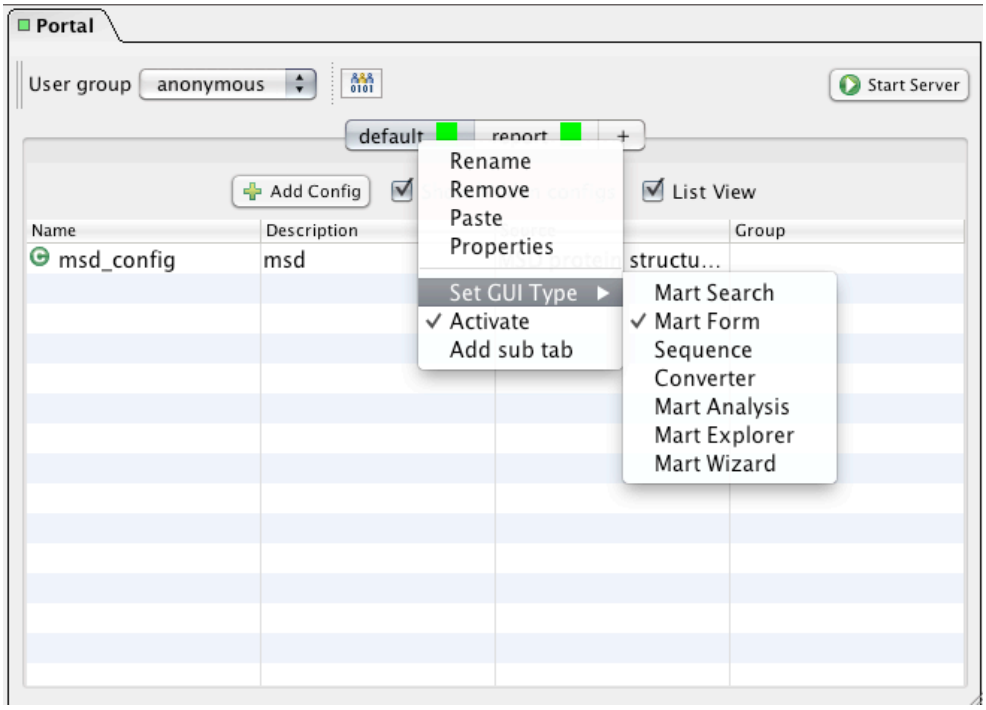
The *Link Index* window will appear:



To create an index, select a link from the *Link* dropdown at the top of the window. Next click on a dataset for which the link is to be created (multiple datasets can be selected using the shift key), and the click the *Create* button.

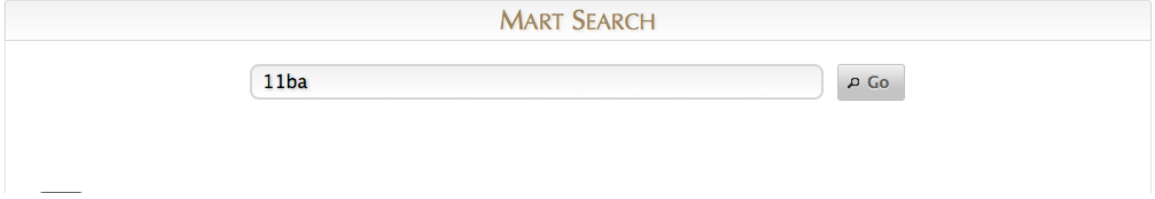
5.12 Changing the GUI type for a config

All access points within a GUI tab must have the same GUI type in the interface. To change this type, right click on the GUI tab you want to modify, select *Set GUI type*, and choose the GUI type from the list

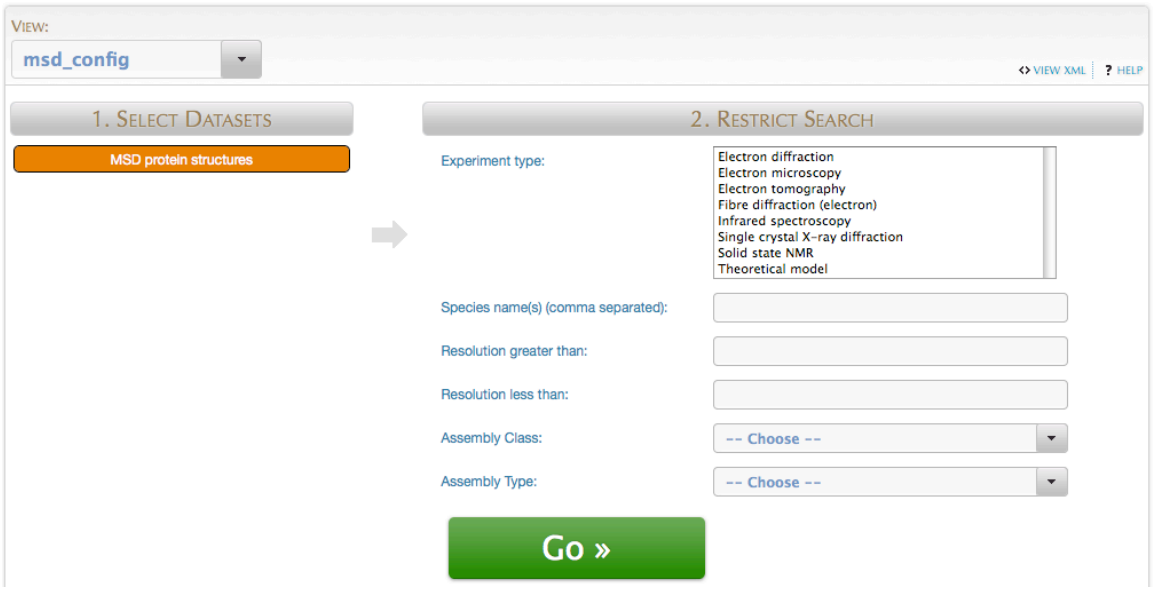


There are four different GUI types supported in the current release: MartAnalysis, MartForm, MartWizard and MartExplorer. (Three other GUI types shown above are currently under development.) These GUIs offer differing levels of complexity, so that deployers may make query pages simpler and easier to use, or more flexible and feature-rich, depending on the situation. If you wish to use MartAnalysis, please keep the number of attributes and/or filters in the config as few as possible as large number of attributes and/or filters will result in queries that may not scale. MartSearch is not yet fully supported and will become available in the near future.

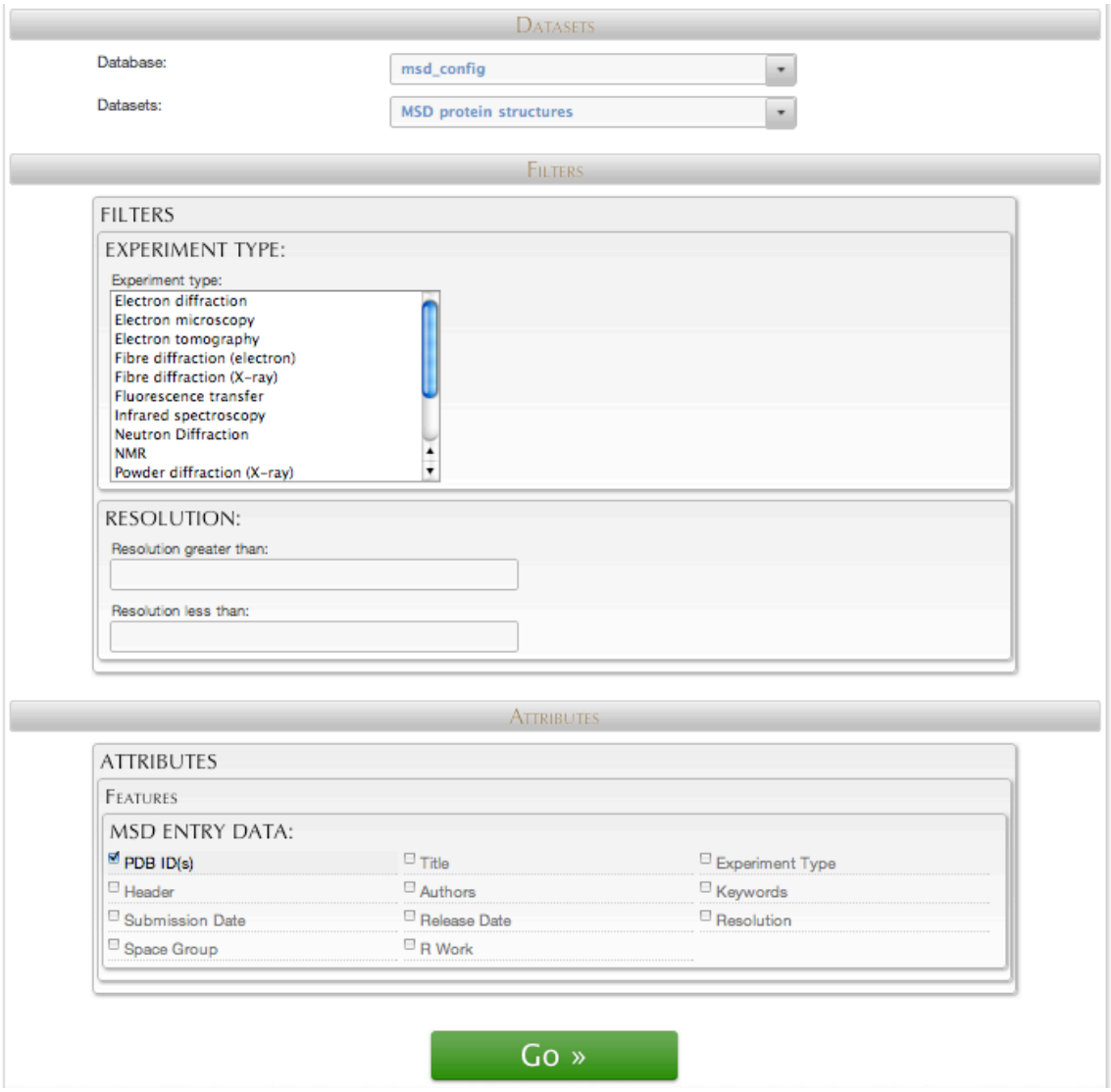
5.12.1. MartSearch



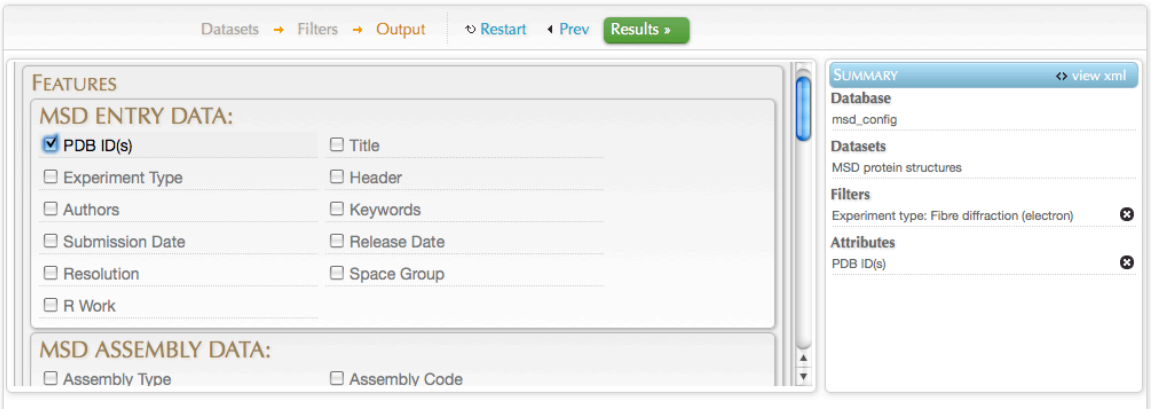
5.12.2. MartAnalysis



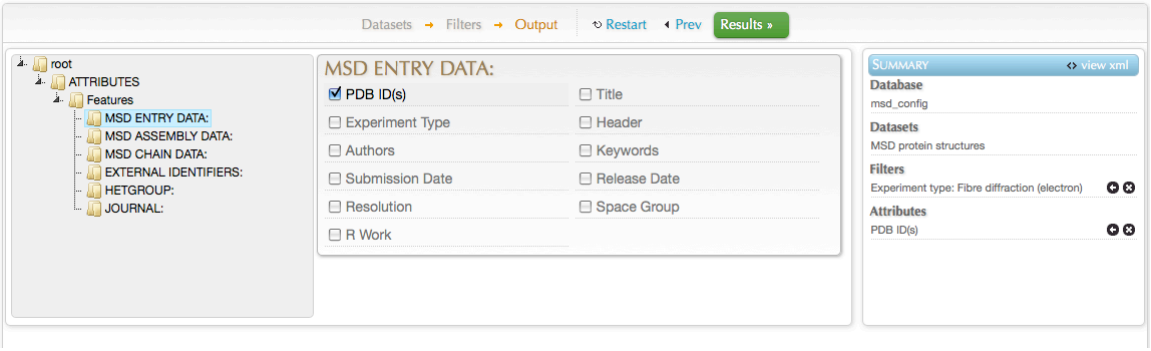
5.12.3. MartForm



5.12.4. MartWizard

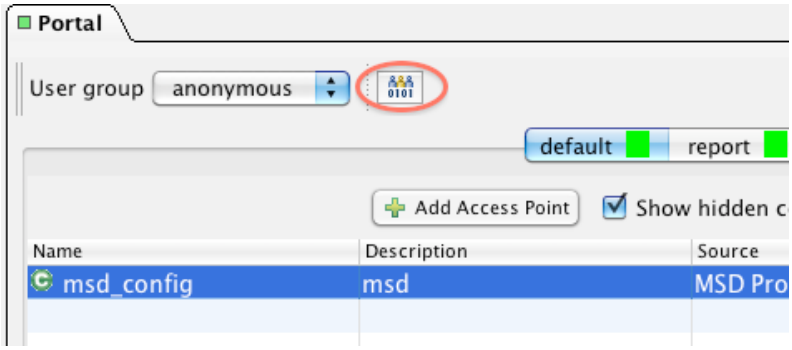


5.12.5. MartExplorer

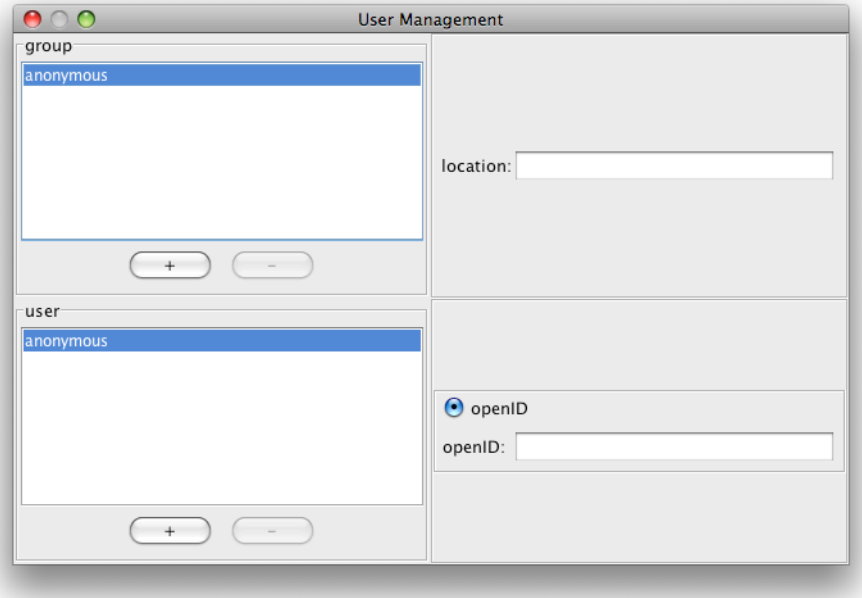


5.13 User management

BioMart supports multi-user access, such that configs can be individually set to be visible or hidden for different groups of users. To manage users and user groups, click on the *user management* icon, located to the right of the user group dropdown at the top of the *Portal* tab.

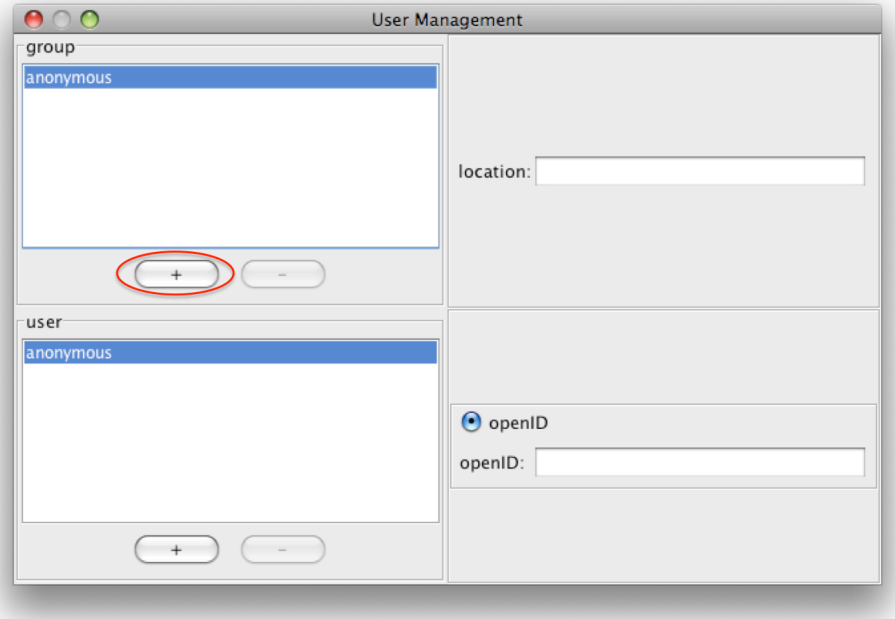


The *User Management* window will appear, showing the current user group and users:



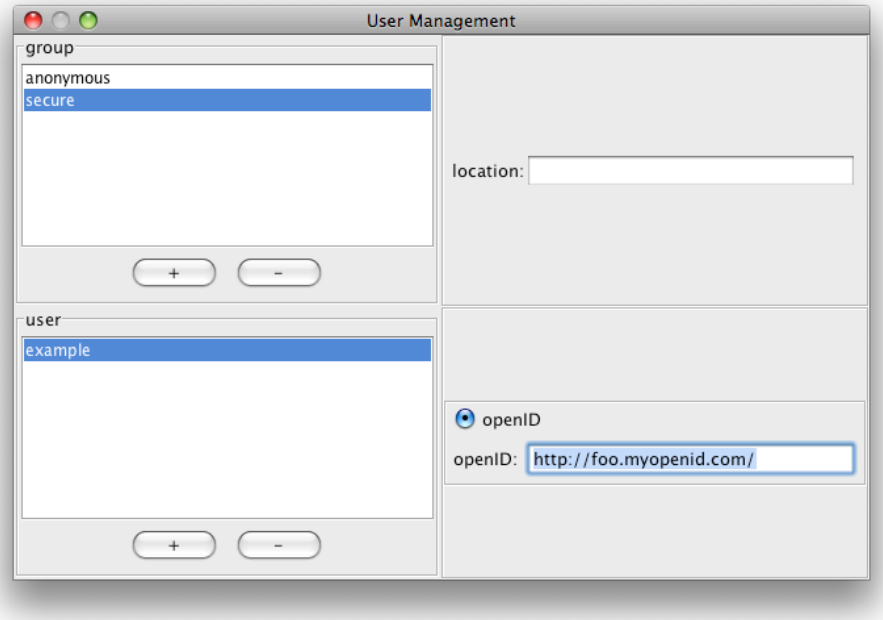
The upper left panel shows the existing user groups, and the lower left panel shows the users in the currently selected group. By default there is one user group containing one user, both called *anonymous*. This user group is used for users who are not logged in.

To add user groups, click the “+” button underneath the *group* panel:



You will be prompted to enter a group name. After entering a name your user group should appear in the list of groups, and a user of the same name will appear in the list of users.

To add a user to a user group, select the group in the upper panel by clicking its name, then click the “+” button underneath the *user* panel and enter a name for the new user when prompted. To allow the user to login you must enter openID credentials in the lower-left corner:

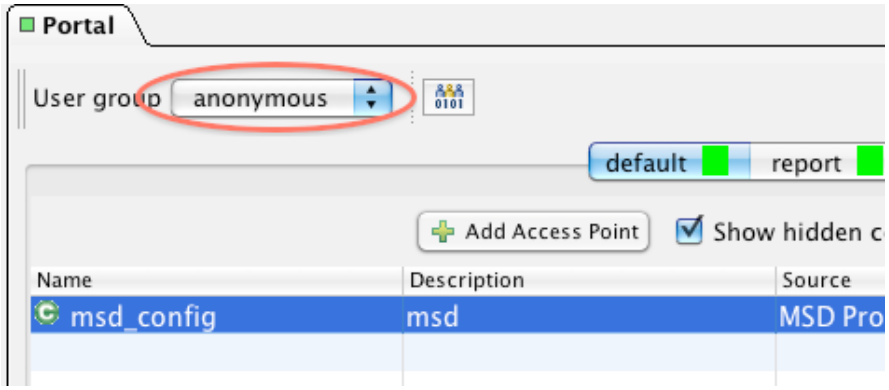


This allows remote authentication. Currently Gmail addresses are supported, as are openID URLs, which may be obtained free of charge via www.myopenid.com. Once configured users can log in by clicking on the link at the top of the deployment website.

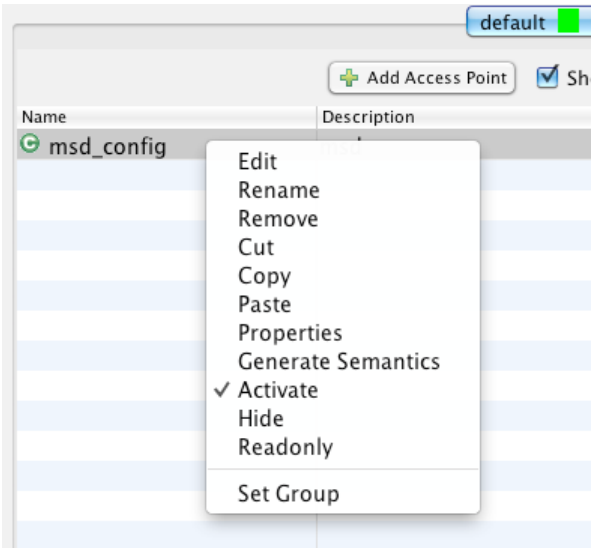
Similarly, to remove a user or group, select it by clicking on its name and then press the “-” button beneath the *user* or *group* panel.

5.14 Hiding an access point

The current user group is shown and can be changed at the top of the *Portal* panel:



To hide a config from users of the current user group, select the access poing, right click on the access point icon and unselect *Activate* such that there is no longer a checkmark next to it. The access point parent GUItab can be inactivated in a similar manner in its right-click menu, and visibility changes to access points under a GUItab cannot be made unless the tab is active.



The green “C” icon will turn grey to indicate that it is inactive for this user group.

NOTE: this only changes the visibility for the current user group. Visibility for each user group must be set independently.

6. Deploy BioMart WebServer

After all the changes have been made, you can click the "Start Server" button on the top right corner to quickly launch BioMart web server. This will also open up your web browser and navigate to BioMart server at http://localhost:9000/. Please be patient, it takes few moments before the BioMart server starts.

Alternatively, to deploy BioMart from command line, from the directory of your installation run the following command:

```
./dist/scripts/biomart-server.sh start
```

To stop the server, please do:

```
./dist/scripts/biomart-server.sh stop
```

It may take several minutes before the server starts up and the site is viewable.

Once server started, please navigate your browser to the proper host and port you just set up, e.g.,

```
http://localhost:9000/
```

7. Configure Deployment

You can also deploy BioMart server from the command line on a server.

In a text editor open the *biomart.properties* file, located in the *dist* subdirectory of the BioMart installation directory.

In the "HTTP settings" section, change the *http.host* property to 0.0.0.0 and change the *http.port* property to the port on which you would like the non-secure web server to run (default 9000). Remove "#" at the beginning of *http.url* line, and set it to the public URL:port from where your BioMart server is accessible, for example:

```
http.url = http://your.domain.org:9000/
```

In the "BioMart Applications Settings" section, change the biomart.registry.file property file to point to "yourfilename.xml" instead of "default.xml" (replacing "yourfilename.xml" with your registry file name). Change the biomart.registry.key.file to point to ".yourfilename" instead of ".default" (again, replacing this with a period followed by your actual registry name, without the ".xml" extension). To start web server, see section 6.

8. Security

8.1 Secure web connections (HTTPS)

If you would like secure access to your BioMart deployment, you will need to further configure your biomart.properties file in the dist subdirectory of the BioMart installation directory. Modify the “HTTPS settings” section as follows:

In the “HTTPS settings” section, remove the “#” at the beginning of the line for the *https.port* property, and change this to the port on which you would like the secure web serve to run (default 9043). Later in the “HTTPS settings” section, remove the “#” from the beginning of the lines for *ssl.keystore* and *ssl.password*. Set the *ssl.password* property to a password of your choice. This password will be used again later in this process.

Next, you will need to have to a valid SSL certificate for the domain that you plan to use to deploy the BioMart server. Here, we demonstrate how to generate the self-assigned SSL certificate. From the root directory of your installation, change to the dist/web/etc directory with the command:

```
cd dist/web/etc
```

If a keystore file exists, delete it using the command:

```
rm keystore
```

Generate a new keystore file using the command:

```
keytool -genkey -keystore keystore -alias biomart -keyalg RSA
```

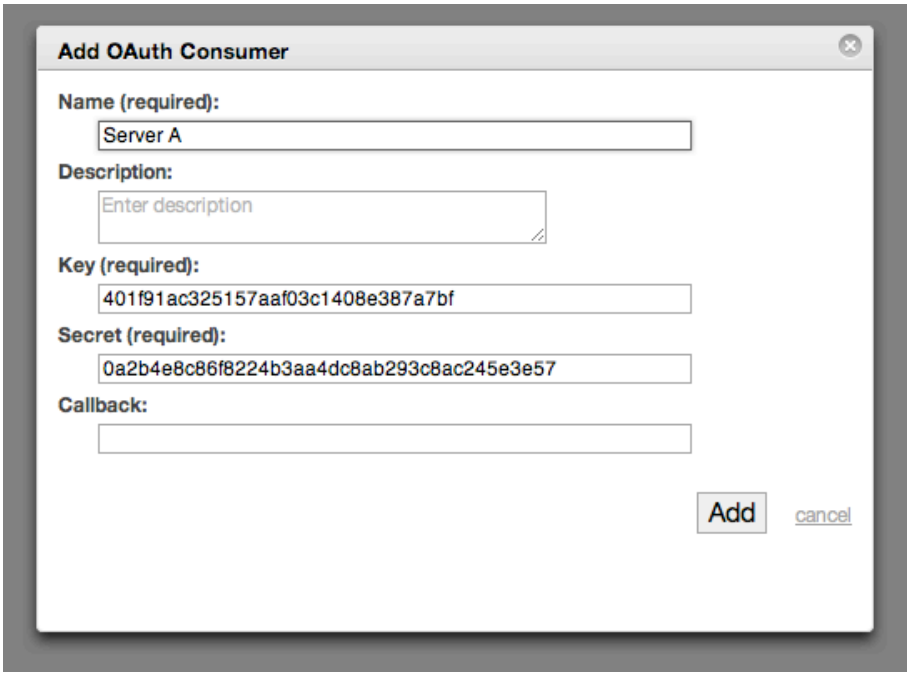
You will be prompted to enter a password; enter the password you set in the ssl.password property in the previous step.

You will then be prompted for your first and last name. Instead, enter in your full server address (e.g. www.yourserver.com). You will then be asked for several more pieces of information; you can leave them all blank. When prompted if the data are all correct, type “yes”. Finally, you will be asked for another password; simply hit “enter” to use the same password you set earlier.

8.2 Authenticated remote access

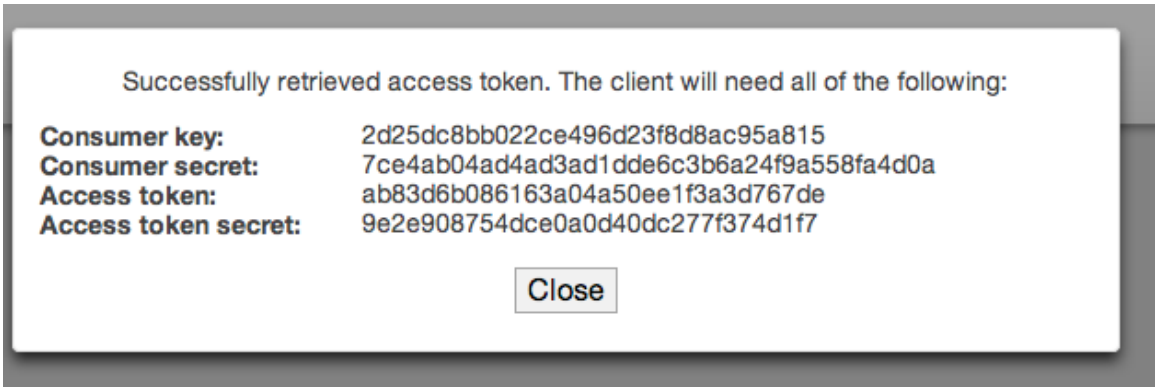
Controlled access for certain authenticated users can be configured in the user management dialogue, described in section 5.13. To enable this

authentication for server-to-server communications, follow the following procedure, which allows “Server A” to securely retrieve data from “Server B.”
On Server B, access the web-based administration interface, located at www.serverB.com/admin (where www.serverB.com is the web address of Server B). You will be prompted for a username and password, which is configured in the file *realm.properties* located in the web/etc subdirectory of the BioMart installation directory. By default, the username is “martadmin” and the password is “P@ssw0rd”.
The administration page has an “Add New Consumer” button, which will allow you to create an access token for Server A.



You can enter any name you like; the important information is the two strings of letters and numbers called “Key” and “Secret.”
You must also authorize the user (or users) by clicking on the “Add access token for user” button and following the on-screen instructions.

Once you have completed the authorization process, a dialogue such as the following will be displayed:

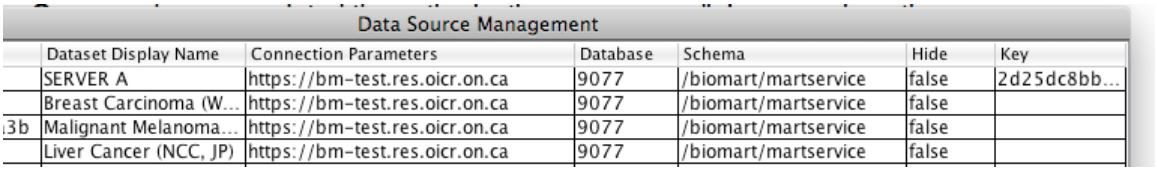


You must combine these four strings, separated by commas. So, in this case, it would be:

2d25dc8bb022ce496d23f8d8ac95a815,7ce4ab04ad4ad3ad1dde6c3b6a24f9a558fa4d0a,ab83d6b086163a04a50ee1f3a3d767de,9e2e908754dce0a0d40dc277f374d1f7

(Note that the line break is not actually present, and is inserted because of the length of the page.) *Be sure not to add any extra spaces or characters after the commas, or it will not work!*

Next, in the registry of Server A in the Data Source Management dialogue, find the connection to Server B (or create it if it does not already exist). In the 'key' column, enter the string you created in the last step.



Server A will now connect securely to Server B, using the credentials of the selected user. You may wish to do the same in the reverse direction if the Server A connects to Server B and you would like it be over secure connection.

9. External Plugins

This section will outline the steps required for writing third-party plugins for BioMart. It is assumed that you already have experience with MartConfigurator and server deployments. This section will assume that your server is deployed locally on http://localhost:9000/. If this is not the case, please replace the URL with the correct address when following this section.

9.1 Creating a new plugin

Create a new directory under the plugins directory to hold your plugin files.

```
cd plugins && mkdir myplugin
cd myplugin
```

Your plugin can host new front-end files to be exposed by the server. Additionally, you can create new GUI types to allow the plugin to be assigned to a GUI container.

9.1.1. Front-end files (client side component)

Create a new directory called **public**.

```
mkdir -p public/myplugin && cd public/myplugin
```

Any files in this directory will be automatically picked up by the BioMart server and can be accessed through a browser under the URL http://localhost:9000/myplugin/

Let's take a basic "Hello World" approach and create an **index.jsp** file under the **public** folder with the following content.

```
<!doctype html>
<%@ page language="java" %>
<%@ page contentType="text/html;charset=UTF-8"%>
<!doctype html>
<html lang="en">
<head>
    <title>Hello World!</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

You can also just use a plain index.html file, but we will use a JSP file so we

can extend it later with more functionality.

Note: You can also put images, CSS, and JavaScript files in the public folder. For the sake of simplicity, we will omit them.

9.1.2. Configure

To let BioMart know about the new plugin, create a new file **guitype.properties** under ***dist/plugins/myplugins*** with the properties displayname and url. The content should look as follows:

```
displayname = My Plugin
url = /myplugin/
```

This will create a new GUI type with **name** = myplugin, **displayName** = My Plugin, and **URL** = /myplugin/.

Note: The URL must match the directory name of the plugin

Note: The URL can include up to two %s strings. When assigned to a GUI container and viewed through the landing page, the first %s is replaced with the GUI container's name, and the second %s is replaced with the MartConfig's name (if applicable). You may need this information for your plugin.

e.g. /myplugin/?gui=%s&mart=%s

You should now have the basic plugin structure ready for deployment.

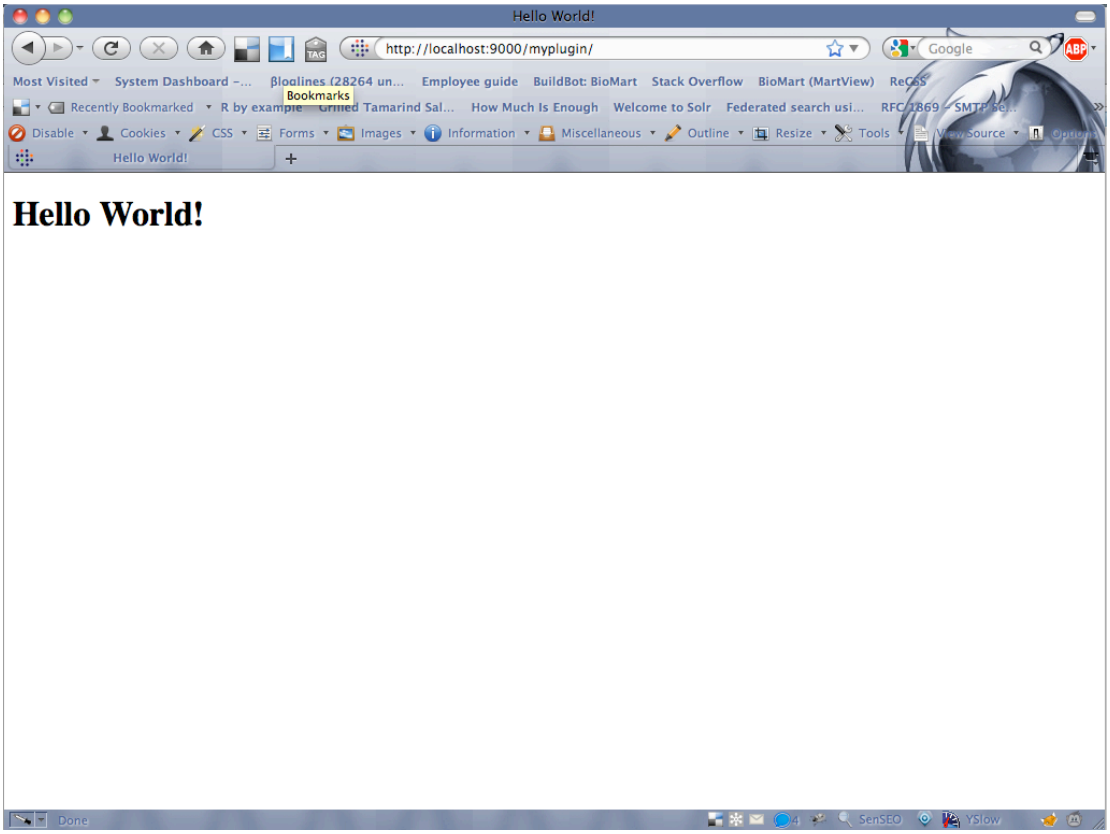
```
dist/
     ...
     plugins/
          myplugin/
               guitype.properties
               public/
                    myplugin/
                         index.jsp
```

9.2 Using the new plugin

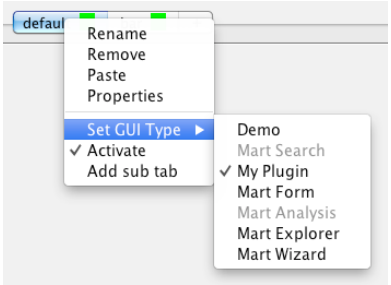
To see your plugin in action, simply deploy the server as normal from the **biomart-java** directory (or from MartConfigurator's **Start Server** button).

```
./dist/scripts/biomart-server.sh start
```

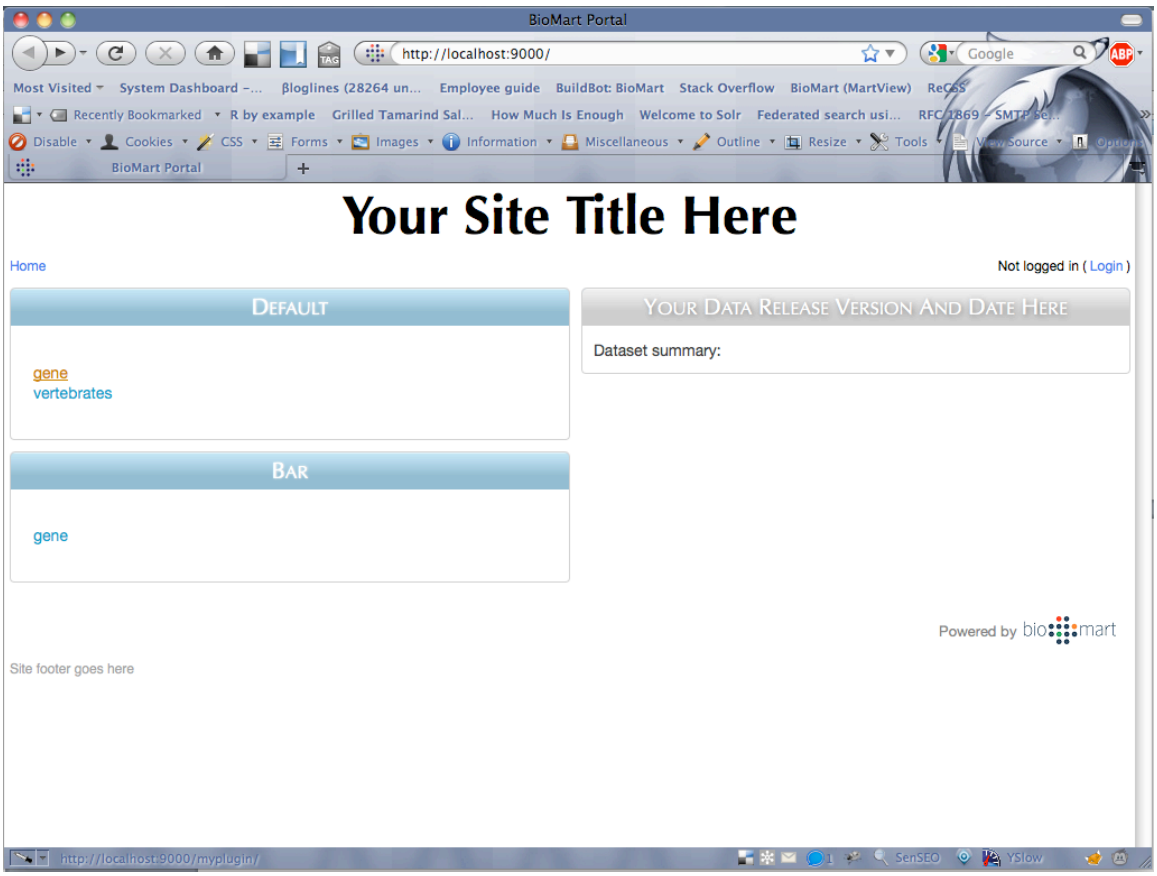
Once the server has started, you can see your new plugin in action by viewing the **URL: http://localhost:9000/myplugin/**



You can assign the plugin to a GUI container through MartConfigurator. From the usual Set GUI Type dialog, select the new **My Plugin** type.



Restart the server and the links under the GUI container with the new type will have the correct **URL (i.e. http://localhost:9000/myplugin/).**



9.3 An advanced example

Go back to the /plugins/myplugin/ directory.

9.3.1. Change plugin URL

We will now append additional information (GUI container and MartConfig) to the end of the URL. Change the content of guitype.properties to the following:

```
displayname = My Plugin
url = /myplugin/?gui=%s&mart=%s
```

9.3.2. Modify index.jsp

We will include the **jQuery** (http://jquery.com/) JavaScript library in order to access some helper methods. Change the content of **index.jsp** to the following:

```
<!doctype html>
<%@ page language="java" %>
<%@ page contentType="text/html;charset=UTF-8"%>
<html lang="en">
<head>
    <title>My Plugin</title>
    <link type="text/css" href="styles.css" rel="stylesheet" />
</head>
<body>
    <h1>My Plugin</h1>
    <div id="gui"><h2>Selected GUI Container</h2></div>
    <div id="mart"><h2>Selected Mart</h2></div>
    <script type="text/javascript"
src="../js/lib/jquery.1.4.2.min.js"></script>
    <script type="text/javascript" src="script.js"></script>
    <script>
      $(initialize);
    </script>
</body>
</html>
```

Note that **$(initialize)** will call the initialize function -- which will be defined later -- when the DOM is ready.

9.3.3. New stylesheet

Create a CSS stylesheet styles.css with the following content:

```
body {
    font-family: Arial, sans-serif;
    font-size: 12px;
    background-color: #eee;
    margin: 10px;
}
h1 { font-size: 24px }
h2 { color: #555 }
#gui, #mart {
    border: 1px dotted #999;
    background-color: #ddd;
    margin: 10px;
    padding: 5px;
}
```

9.3.4. New JavaScript file

Now create a JavaScript file script.js with the following content:

```
function initialize() {
    var queryString = location.search.substr(1),
        parts = queryString.split('&'),
        _gui = parts[0].split('=')[1],
        _mart = parts[1].split('=')[1];


    $.ajax({
        url: '/rest/json/gui/' + _gui,
        success: function(json) {
            var gui = json,
                mart;

            for (var i=0; mart=gui.marts[i]; i++) {
                if (mart.name == _mart) {
                    break;
                }
            }

            $('#gui')
                .append('<p><strong>GUI:</strong> '
                    + gui.displayName + '</p>')
                .append('<p><strong>Number of marts:</strong> '
                    + gui.marts.length + '</p>');

            $('#mart')
                .append('<p><strong>Mart:</strong> '
                    + mart.displayName + '</p>')
                .append('<p><strong>Operation:</strong> '
                    + mart.operation + '</p>');
        }
    });
}
```

Note: Make sure that the ***script.js*** and ***styles.css*** files are within the same directory as the ***index.jsp*** file

The updated directory structure would look like this:

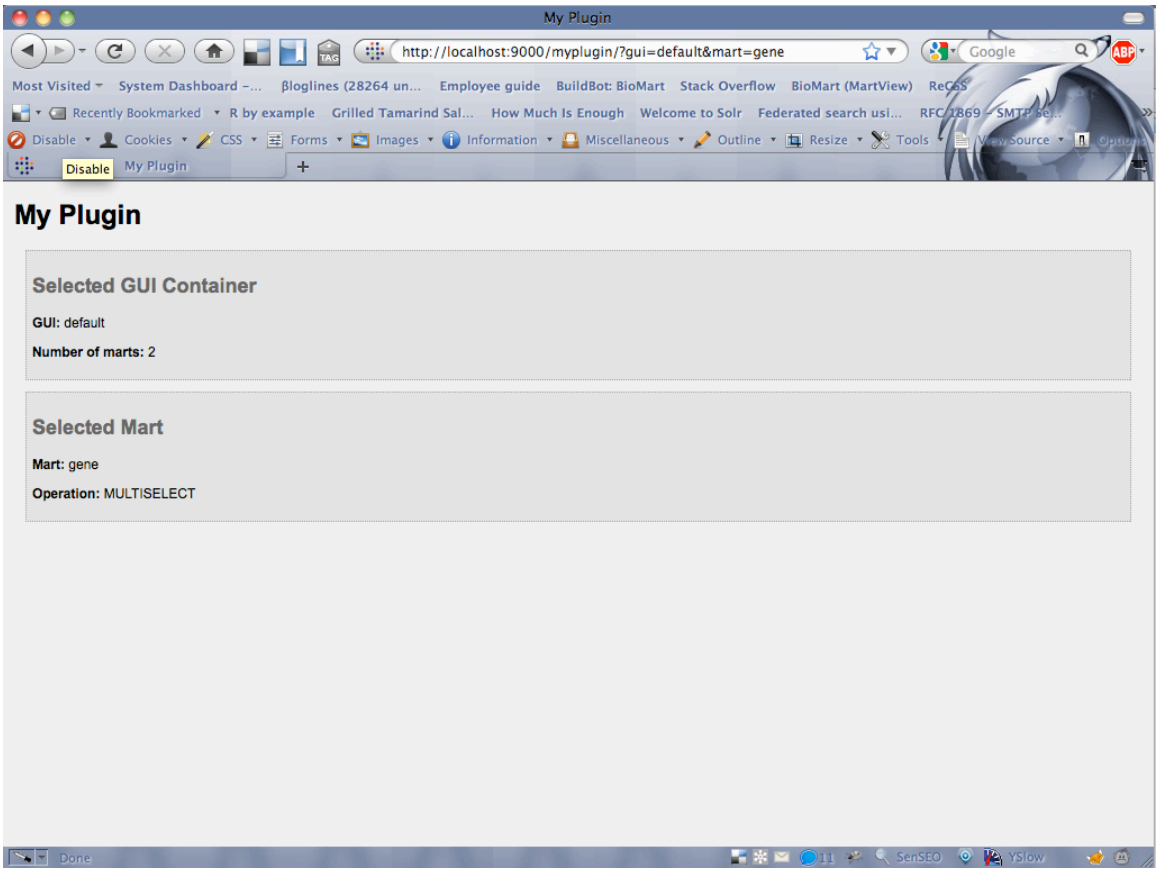
```
dist/
     ...
     plugins/
          myplugin/
               guitype.properties
               public/
                    myplugin/
                         index.jsp
                         script.js
                         styles.css
```

9.3.5. Redeploy

Restart the server so it can pick up the new URL structure.

```
./dist/scripts/biomart-server.sh restart
```

Now a link to the plugin page should look similarly to the screenshot below.



The JavaScript make a MartService API call to fetch the GUI container information. This example demonstrates how to interactive with the MartService API within your plugin.

9.4 Custom Processors (server side component)

A plugin may include one or more custom processors. A processor is used to return query results in different formats. By default BioMart comes with the TSV (tab separated value) processor.

To create a new processor, you will need to create a new Java class in the `org.biomart.processors` package that implements

`ProcessorInterface` in the same package. The name of the new processor class is the same name that is used in the Query XML.

The easiest way to create this class is using a Java IDE such as Eclipse, NetBeans, or IntelliJ. In the IDE, create a new Java project, and add the **biomart-0.8.jar** file from the **dist/lib** directory to the project's Library.

Add a new package named `org.biomart.processors`, and create your processor class inside.

The processor class has to implement the `public void printResults(QueryRunner qr, OutputStream outputHandle)` method. At minimum, you will need to set the output handler of the `QueryRunner` instance.

For example, the TSV processor simply sets the `outputHandle` from the method's argument as the output handle of the `QueryRunner` instance.

```
// TSV.java
package org.biomart.processors;
import java.io.IOException;
import java.io.OutputStream;
import java.sql.SQLException;
import org.biomart.common.exceptions.TechnicalException;
import org.biomart.queryEngine.QueryRunner;

public class TSV implements ProcessorInterface {
    @Override
    public void printResults(QueryRunner qr,
            OutputStream outputHandle) throws TechnicalException,
            SQLException, IOException  {

        QR.setOutputHandle(outputHandle);
        try {
            QR.runQuery();
        } catch (InterruptedException ex) {
            // ...
        }
    }
}
```

The default output format is TSV (columns separated by tabs, rows separated by newline characters). In order to change the output format, you will need to either implement your own `java.io.OutputStream` that extends `java.io.FilteredOutputStream`, or wait until all results come back from the `outputHandle` and post-process it before writing to the output handle of the `QueryRunner` instance.

When you are finished with the implementation, use the IDE to build your project to a JAR file.

Create a directory under **plugins** that holds a single directory called **lib**. Copy your project's JAR file to this **lib** directory and BioMart will install it automatically.

Please see the CSV (comma separated value) processor included in the BioMart SVN repo (**plugins/csvprocessor**). The source file included (CSV.java) shows an example where a custom OutputStream is created.

10. Extending Query Handling

10.1 Introduction

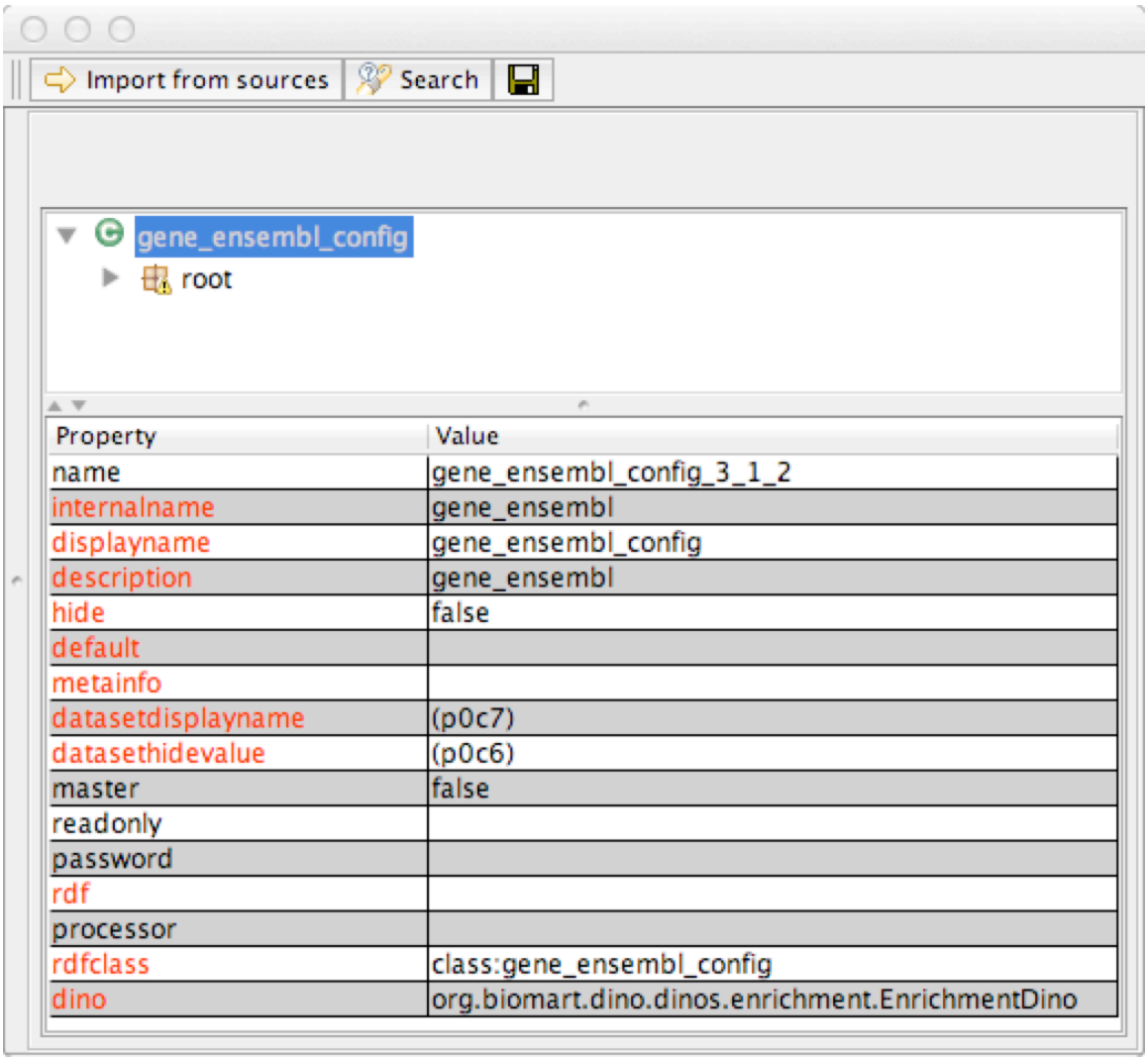
In BioMart it's possible to modify how queries are handled through its extension system. After an extension is attached to a configuration, all the query requests involving that configuration submitted either programmatically, through web services or Java API, or from the web graphical interfaces will be handled by the extension.

An extension can express requirements that will be bound to the filters and attributes of the query currently processed in a declarative manner directly inside it's source code. How to exactly bound filters and attributes to the requirements is specified by the deployer with MartConfigurator.

An extension is called **Dino**.

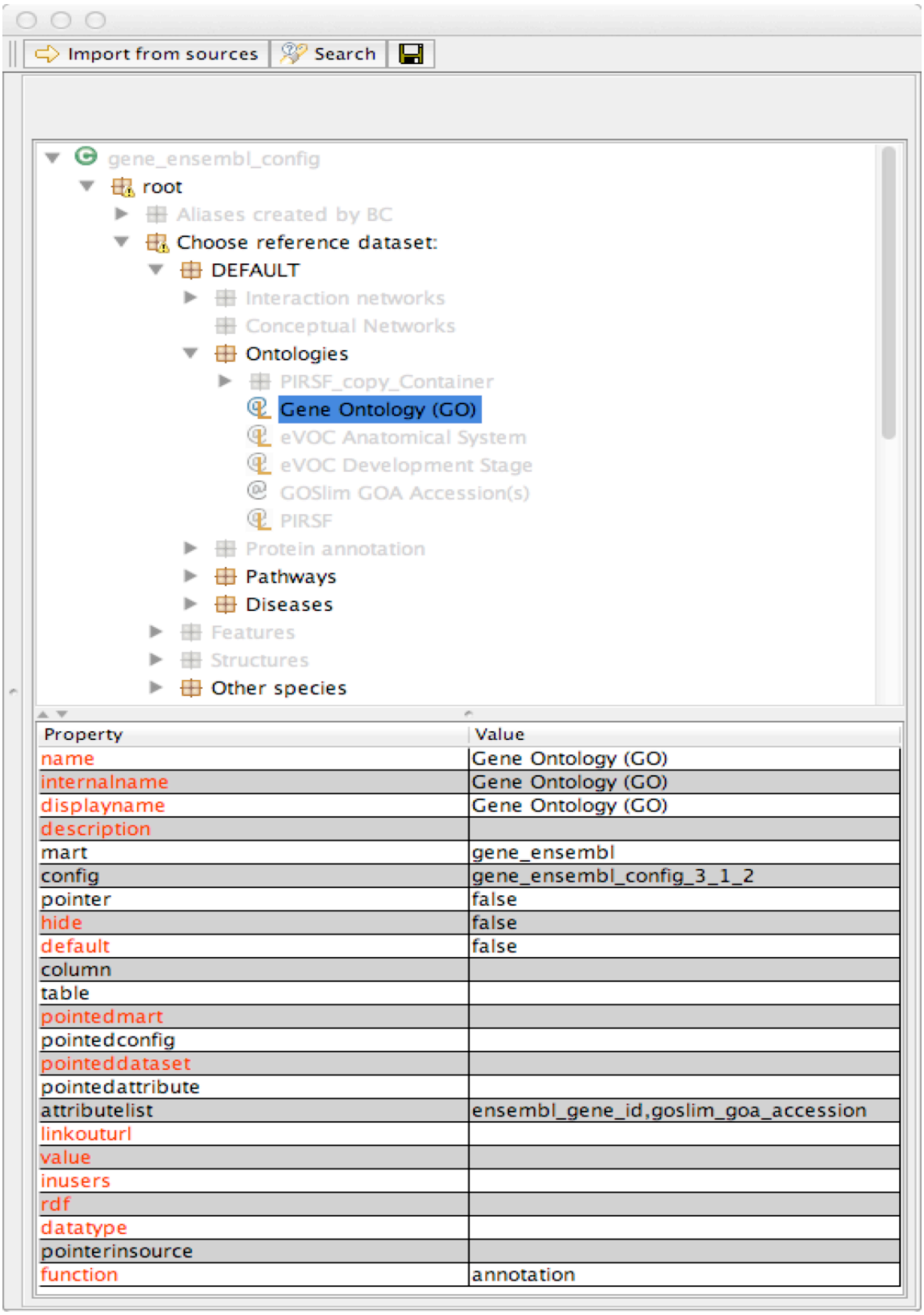
10.2 Configuration

To specify a dino for a configuration, from MartConfigurator click on a configuration and fill the *dino* property with the name and namespace of the dino class

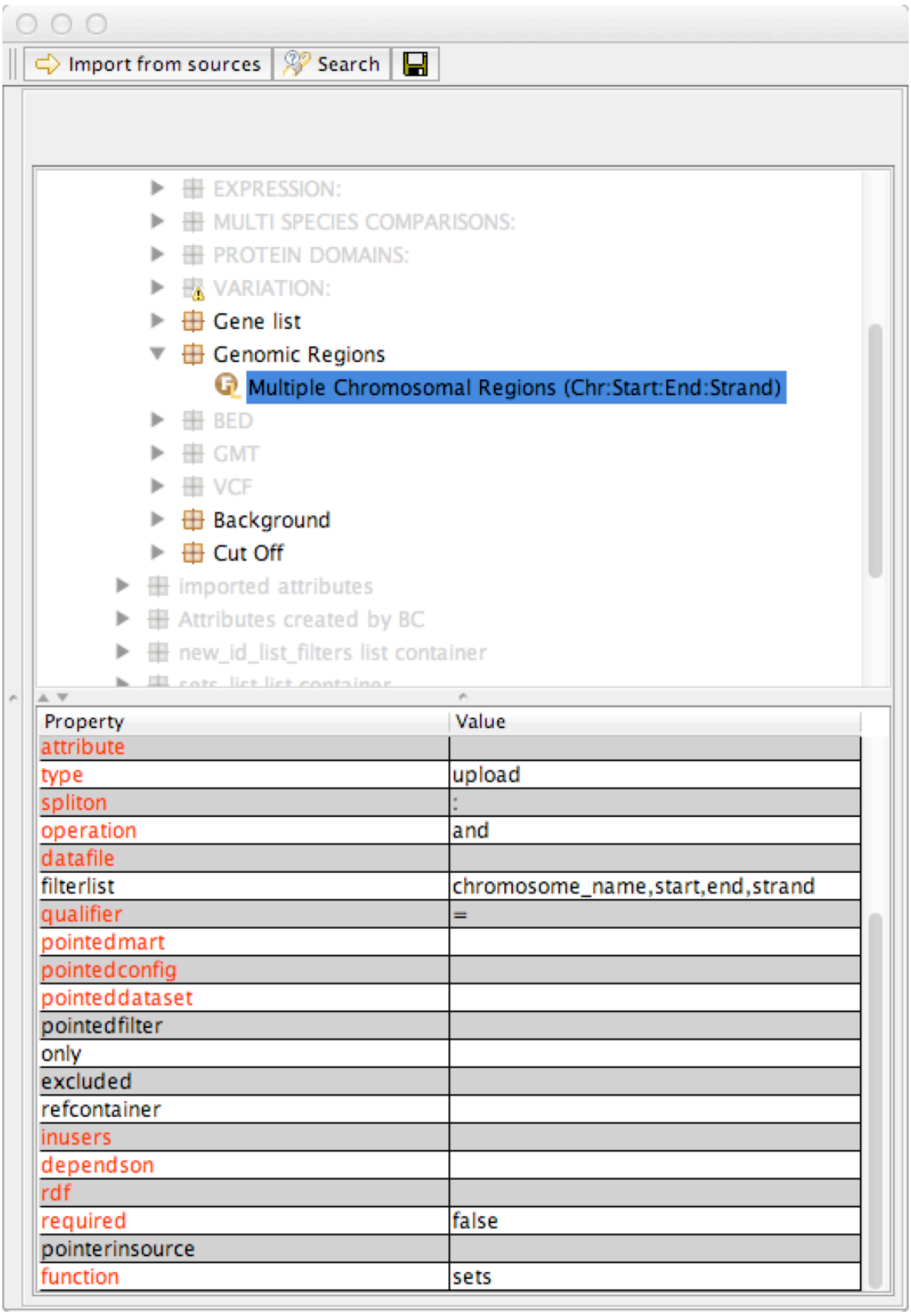


Then, it must be marked which filter and attribute will be bound to the input for the dino, and this is accomplished filling the *function* property of elements.

In the following example we're saying that the *Multiple Chromosomal Regions* filter **value** within a query will become the *sets* input, and the *Gene Ontology* attribute **name** will be become the *annotation* input for the dino.



Property	Value
name	Gene Ontology (GO)
internalname	Gene Ontology (GO)
displayname	Gene Ontology (GO)
description	
mart	gene_ensembl
config	gene_ensembl_config_3_1_2
pointer	false
hide	false
default	false
column	
table	
pointedmart	
pointedconfig	
pointeddataset	
pointedattribute	
attributelist	ensembl_gene_id,goslim_goa_accession
linkouturl	
value	
inusers	
rdf	
datatype	
pointerinsource	
function	annotation


Import from sources
Search
EXPRESSION:
MULTI SPECIES COMPARISONS:
PROTEIN DOMAINS:
VARIATION:
Gene list
Genomic Regions
Multiple Chromosomal Regions (Chr:Start:End:Strand)
BED
GMT
VCF
Background
Cut Off
imported attributes
Attributes created by BC
new_id_list_filters list container
Property
Value
attribute
type
upload
spliton
:
operation
and
datafile
filterlist
chromosome_name,start,end,strand
qualifier
=
pointedmart
pointedconfig
pointeddataset
pointedfilter
only
excluded
refcontainer
inusers
dependson
rdf
required
false
pointerinsource
function
sets

Note that the function assigned to a filter list or an attribute list will be inherited by all the filters and attributes, respectively, of that list.

10.3 Implementing a Dino

A dino is a Java class that implements the Dino interface defined under the *org.biomart.dino.dino* namespace. There are few methods that must be implemented and they can be found in the Dino interface file.

As said before, a dino expresses its requirements, that is the input for each query, in a declarative way. This is done through the *@Func* Java annotation defined under the *org.biomart.dino.annotations* namespace that has two attributes: *id*, *optional*.

Example:

```
3  public class Converter implements Dino {
4      @Func(id = "genes") String source,
5      @Func(id = "convert to") convertTo,
6      @Func(id = "annotation", optional = true) annotation;
7
```

In this example we created a *Converter* dino stating that the query must contain two elements with *function* property value equal to *genes* and *convert to*, and a third one, optional, with property value *annotation*.

At the moment that a new query, with same configuration as the one the Converter dino has been attached to comes in, a new instance of the class is created, provided with the *Query* object and run.

The binding of the query element values to the state of the dino (in this case, source, convertTo and annotation) doesn't happen automatically but the dino instance itself does it using the Binding utility class. In this way a dino has greater control over the binding and can bind element values to its state variables multiple times per request. Follows an example,

```
    @Override
    public Dino setQuery(Query query) {
        this.q = query;
        return this;
    }

    @Override
    public void run(OutputStream out) throws Exception {
        List<Field> myFields = Binding.getAnnotatedFields(this.getClass());
        List<QueryElement> elements = this.q.getQueryElements();
        List<QueryElement> boundEls = Binding.setFieldValues(this, myFields, elements);
        // creates the binding table
        // Key: String - function id
        // Value: QueryElement | Element - the element
        this.metaData.setBindings(myFields, boundEls);

        doRun();
    }
```

10.3.1. Dependency Injection

Google Guice is used for dependency injection and the module that defines injectables for dinos is under the *org.biomart.dino* namespace and is called *DinoModule* (surprise!).

10.4 Enrichment

Note that functionality could change in the future.

The Enrichment Dino is an extension that given a set, cutoff, an optional background and other requirements, conducts an enrichment process using a C program written by Prof. Paolo Provero of University of Turin and returns the results with format as specified by the *processor* query parameter if the *header* parameter is equal to *false*, as json otherwise.

To use this extension, the value of the *dino* property for the configuration must be set to org.biomart.dino.dinos.enrichment.EnrichmentDino.

The function property **values** to set on filters and attributes of a configuration with MartConfigurator for this dino are

- *sets* – the filter that holds the gene list.
- *background* (optional) – the filter that holds the gene list acting as background. In case of a query without background the whole genome will be used.
- *annotation* – the attribute that represents the annotation to use. There could be multiple attributes with this function per query.
- *bonferroni* – a Boolean filter that states whether compare the cutoff with corrected p-value or not.
- *gene_limit* (optional) – this should be a filter of type singleSelectBoolean that represents the category of genes to filter the results.
- *homolog* (optional) –

Note that species translation is supported.

This extension though needs further information to work: within the dist/conf/dinos/enrichment/EnrichmentDino.json file, as you can see from the template file already present, it must be specified a table with parameters used for format translation and getting information to build the response for the web GUI.

- *gene* – a table of parameters to use to format translation and information retrieval that must contain

    - *gene_attribute* – attribute to use for gene output format translation.
    - *description_attribute* – attribute to use to get the description of a gene.
    - *other_attributes* (optional) – list of attributes to use to get other information to build the response for the web GUI.
- *annotation* – a table where each entry has as key the **internal** name of an attribute used as *annotation* and as value a table like this
    - *description_attribute* – attribute to use to get the description of an annotation.
    - *other_attributes* – list of attributes to use to get other information to build the response for the web GUI.

10.4.1. BED format, downstream, upstream support

Note that this functionality could change in the future.

If you want to support the BED file format, upstream, downstream for the set, you need to set the *dino* property to org.biomart.dino.dinos.EnrichmentDecorator and the function property **values** to set on filters and attributes of a configuration with MartConfigurator for this dino are

- bed_regions – the filter that holds the BED file content
- downstram – the filter that holds the downstream value
- upstream – the filter that holds the upstream value

And finally change the dist/conf/dinos/enrichment/EnrichmentDecorator.properties file content to

- *enrichment_decorator.bed_to_ensembl_output_filter* – the *name* of the filter with *function* property equal to *regions*. This filter must be of the same type of the *bed_regions* filter.
- *enrichment_decorator.upstream_downstream_output_filter* – the *name* of the filter with *function* property equal to *sets*. This filter must be of the same type of the *bed_regions* filter.

11. Preconfigured Portal Deployment

11.1 ICGC

11.1.1. Changing connection parameters

For a local deployment of ICGC, you will want to point the configuration to your local database. To simplify the update process, a new command line update tool is developed. To update the included icgc5.xml with your connection parameters, you must first create a configuration file that includes those connection parameters. Copy and paste the following text to a text file, making the appropriate changes to the connection parameters to point to your database (i.e. YOURSERVER becomes your database server address; YOURDBNAME is where the ICGC data is loaded to by martloader tool; contact ICGC DCC at *dcc-support@lists.oicr.on.ca* for your DATASETNAME etc.):

```
<config>
<oauth openid="https://me.yahoo.com/a/IyINRct235wr0Il0NtQMUcKwenvgzIUqbyaA"/>
<datasets>
<dataset name="DATASETNAME" database="YOURDBNAME" dbhost="YOURSERVER"
dbport="YOURPORT" dbuser="YOURUSER" password="YOURPASSWORD"/>
</datasets>
</config>
```

Save this file as *update.xml* in the *dist* subdirectory of your BioMart installation directory. Then, from the *dist* subdirectory, run the following command:

```
./scripts/updateICGC.sh ../registry/icgc5.xml update.xml
```

A new portal configuration file with a time stamp as part of its name, something like: *icgc5-2011-56-07-17-56-59.xml,* will now be generated with updated information to point to your local database. A key file (named as *.icgc5-2011-56-07-17-56-59*) will be generated too. This key file is used to encrypt/decrypt the passwords in the xml file, so you can share the xml file with others without worrying about exposing password.

11.1.2. Quick deployment

Once previous step is done, you can quickly deploy the BioMart server with your data using default settings by:

```
./scripts/biomart-server.sh start
```

This will create an instance of the BioMart web tool on your local machine. You can access it manually by navigating to http://localhost:9000/ in a browser window.

Once your test instance is deployed, please test that querying returns results for both your dataset(s) and for datasets from other ICGC members. For advance deployment please refer to 10.1.6.

11.1.3. Configuring site appearance

The BioMart installation directory contains a directory named *themes* in which sets of pre-configured site web pages (header, sidebar, footer, etc.) are provided. An *icgc5* theme is included with the current BioMart release.

To enable a theme for your ICGC portal site --- *before you've run 'ant' to build BioMart* --- edit the file *ant.properties* in the BioMart installation directory and change the property 'theme' from *default* to your theme of choice:

theme = icgc5

After performing the BioMart build, the *dist/web/includes/* subdirectory of your BioMart installation directory will contain a file *header.html* that you are encouraged to edit to include site logos, etc. specific to your institution. To maintain a consistent look and feel across ICGC portal installations, users should avoid making unnecessary modifications to any other files included in the ICGC theme directory.

11.1.4. Configuring the location dropdown

To enable the locations dropdown on the homepage and add your site, you must open the file *locations.properties* in the *dist/web/etc/* subdirectory of your BioMart installation directory. In that file, add the line

location.code = *yourcode*

Where *yourcode* is replaced with your three-digit ICGC institution code. Contact *dcc-support@lists.oicr.on.ca* if you are an ICGC member and do not know your institution code.

11.1.5. Security

To password protect your entire site while testing, you must first edit the file *web.xml* located in the *dist/web/webapps/martapps/WEB-INF/* subdirectory of your BioMart installation directory. Find the line that reads

<!-- Uncomment to enable site-wide BASIC authentication

(Line number 14) and delete this entire line. Ten lines later there is a line that reads

-->

Delete this line as well, and save the file.

By default, this will protect your site with the username “martuser” and the password “123456”. To change the username and/or password, edit the file *realm.properties* in the *dist/web/etc/* subdirectory. Change the value “martuser” to whatever username you would like, and the value “123456” to whatever password you would like. You can also add additional users by adding new lines to the file, in the format

username: *password*,user

Note that the “,user” at the end of the line is mandatory.

For deployment servers, you MUST change the martadmin password in the realm.properties file.

To enable HTTPS protocol so that communication between your server and all clients is encrypted, please see section 8.1.

11.1.6. Advanced deployment

For a more configurable deployment (e.g. to set the port for your server) you must deploy the server from the command line. To do so, you must edit the *biomart.properties* file, located in the *dist* subdirectory of the BioMart installation directory.

In the "HTTP settings" section, change the *http.host* property to 0.0.0.0 and change the *http.port* property to the port on which you would like the non-secure web server to run (default 9000, for production server use port 80). Remove “#” at the beginning of *http.url* line, and set it to the public URL:port from where your BioMart server is accessible, for example:

```
http.url = http://your.domain.org:9000/
```

In the "BioMart Applications Settings" section, change the biomart.registry.file property file to point to the xml file generated in section 10.1.1. Change the biomart.registry.key.file to point to the key file generated in section 10.1.1. (note that it begins with a period). You may

need to copy these files from the machine where you configured to the server on which you intend to deploy if the two they are not the same machine. Save your changes to the properties file before launching the server.

To start the server, from your BioMart install directory run the following command:

```
./dist/scripts/biomart-server.sh start
```

It may take several minutes before the server starts and the site is viewable.

To stop the server, enter:

```
./dist/scripts/biomart-server.sh stop
```

12. BioMart API

The BioMart API Consists of three parts: REST API, SOAP API, and Java API. All three APIs have access to the same methods, so you can choose the one you are most comfortable with.

12.1 Query XML

The XML's root element must be <Query/>, and must specify these attributes:

- client: name of the client making the call (choose something appropriate)
- processor: any valid processor name (e.g. TSV)
- limit: the number of rows to return (-1 for no limit)
- header: if set to 1 then first row of results will be column headers

The <Query/> element can contain one or more <Dataset/> elements with the following attributes:

- name: the name of the dataset
- config (optional): the config of the mart (if applicable)

Each <Dataset/> element may contain zero or many <Filter/> elements:

- name: name of filter
- value: value of filter Each <Dataset/> element must contain at least one <Attribute/> element:
- name: name of attribute

```
Sample Query XML
<Query client="" processor="TSV" limit="1000" header="1">
<Dataset name="hsapiens_gene_ensembl_hopkinsBreast3"
config="hsapiens_gene_ensembl">
<Filter name="biotype" value="miRNA"/> <Attribute
name="ensembl_gene_id"/> <Attribute name="cancertype"/>
<Attribute name="description"/> <Attribute
name="start_position"/> <Attribute name="end_position"/>
</Dataset> </Query>
```

12.2 Processors

By default, results from a Query are returned in TSV (tab-separated) format. To change the format, simply change the processor type in the Query XML.

The following core processors are available in BioMart:

TSV

Returns tabular data with attributes separated by a tab character, and rows separated by a newline character.

Example:

```
Attribute1    Attribute2    Attribute
value1a       value2a       value3a
value1b       value2b       value3b
value1c       value2c       value3c
...
```

CSV

Returns tabular data with attributes separated by a comma (,) and wrapped around double-quotes (”), and rows separated by a newline character.

Example:

```
"Attribute1","Attribute2","Attribute3
"value1a","value2a","value3a"
"value1b","value2b","value3b"
"value1c","value2c","value3c"
...
```

JSON

Returns a JSON object containing the results and the total number of rows found.

Note: The results are not streamed to the client when using the JSON processor. This means that data will be written only when the entire result set is ready, which may not be ideal for slower queries.

Example:

```
{
    "total": 100,
    "data": [
        {
            "Attribute1": "value1a",
            "Attribute2": "value2a",
            "Attribute3": "value3a"
```

```
        },
        {
            "Attribute1": "value1b",
            "Attribute2": "value2b",
            "Attribute3": "value3b"
        },
        {
            "Attribute1": "value1c",
            "Attribute2": "value2c",
            "Attribute3": "value3c"
        },
        ...
    ]
}
```

12.3 REST API

The REST API is built on top of hte Java API for RESTful access. This API supports two response formats: XML and JSON. Both formats return the same data in different serialized representations.

The following are two different representations of the same Mart.

XML Representation (Default)

```
<mart
    description="gene_vega_config"
    displayName="gene_vega_config"
    name="gene_vega_config"
    config="gene_vega_config"
    isHidden="false"
    meta=""
    operation="MULTISELECT"
/>
```

JSON Representation

```
{
    "isHidden": false,
    "displayName": "gene_vega_config",
    "config": "gene_vega_config",
    "operation": "MULTISELECT",
    "meta": "",
    "name": "gene_vega_config",
    "description": "gene_vega_config"
}
```

12.3.1. Specifying Response Format

There are two ways to specify the response format.

Adding the corresponding HTTP Accept header into the request.

- application/xml for XML
- application/json for JSON

Appending the request with the corresponding file extension

- .xml for XML
- .json for JSON

For Example, the following both return the response in JSON format.

- curl -H "Accept: application/json" http://dcc.icgc.org/martservice/marts
- curl http://dcc.icgc.org/martservice/marts.json

Note: The default format is XML, so the method http://dcc.icgc.org/martservice/marts will return XML unless the Accept header specifies otherwise.

12.3.2. Status Codes

The following status codes are returned from the REST API.

- **200 OK**: Success!
- **400 Bad Request**: The request was invalid. The accompanying message will explain why.
- **404 Not Found**: The requested resource was not found.
- **405 Method Not Allowed**: The method was invalid for this request. (e.g. Using POST for a GET request)
- **500 Server Error**: Something is broken. Please send a message to the mailing list (users@biomart.org) with the request and any accompanying response message.

12.3.3. Resources

The following is a list of resources available from the BioMart REST API.

12.3.3.1. Portal

Returns the root GUI Container containing all child Containers and associated Marts.

URL
/martservice/portal

Method

GET

Parameters
guitype (optional) - Restricts returned containers to only the specified GUI type (e.g. martform)

Returns
Root GUI Container object

Sample URL
/martservice/portal?guitype=martform

Sample XML Response

```
<guiContainer guiType="" isHidden="false" description=""
displayName="root"
    <guiContainers>
            name="root">
        <guiContainer guiType="martform" isHidden="false"
                description="default"
                displayName="default" name="default">
            <marts>
                <mart config="snp_config" isHidden="false"
                        meta=""
                        operation="MULTISELECT"
                        description="snp_config"
                        displayName="snp_config"
                        name="snp_config"/>
                ...
            </marts>
        </guiContainer>
        ...
    </guiContainers>
</guiContainer>
```

Sample JSON Response

```
{
    "isHidden": false,
    "guiContainers": [
        {
            "isHidden": false,
            "guiContainers": [ ],
            "marts": [
                {
                    "isHidden": false,
                    "displayName": "snp_config",
                    "config": "snp_config",
                    "operation": "MULTISELECT",
                    "meta": "",
                    "name": "snp_config",
                    "description": "snp_config"
                },
```

```
                ...
            ],
            "guiType": "martform",
            "name": "default",
            "displayName": "default",
            "description": "default"
        },
        ...
    ],
    "marts": [ ],
    "guiType": "",
    "name": "root",
    "displayName": "root",
    "description": ""
}
```

12.3.3.2. Marts

Lists all the Marts available from the server.

URL
/martservice/marts

Method
GET

Parameters
guicontainer (optional) - Restricts the returned Marts to the specified GUI Container

Returns
List of Mart objects

Sample URL
/martservice/marts

Sample XML Response

```
<marts>
    <mart config="snp_config" isHidden="false" meta=""
operation="MULTISELECT"
            description="snp_config" displayName="snp_config"
name="snp_config"/>
</marts>
```

Sample JSON Response

```
[
    {
```

```
        "isHidden": false,
        "displayName": "snp_config",
        "config": "snp_config",
        "operation": "MULTISELECT",
        "meta": "",
        "name": "snp_config",
        "description": "snp_config"
    },
    ...
]
```

12.3.3.3. Datasets

Lists all the Datasets for the given Mart.

URL
/martservice/datasets

Method
GET

Parameters
config (required) - The name of the Mart to return datasets from

Returns
List of Dataset objects

Sample URL
/martservice/datasets?config=snp_config

Sample XML Response

```
<datasets>
    <dataset isHidden="false" description="btaurus_snp"
            displayName="Bos taurus Variation (dbSNP 130)"
name="btaurus_snp"/>
    ...
</datasets>
```

Sample JSON Response

```
[
    {
        "isHidden": false,
        "name": "btaurus_snp",
        "displayName": "Bos taurus Variation (dbSNP 130)",
        "description": "btaurus_snp"
    },
    ...
```

```
]
```

12.3.3.4. Filters

Lists all the filters for the given Dataset(s).

URL
/martservice/filters

Method
GET

Parameters
datasets (required) - Comma-separated string of Datasets.

config (optional) - The name of the config as returned by the Mart object. You should provide this when the Mart's config is non-empty.

container (optional) - The name of the container you want to return filters for.

Returns
List of filter objects

Sample URL
/martservice/filters?datasets=btaurus_snp&config=snp_config

Sample XML Response

```
<filters>
    <filter isHidden="false" qualifier="" type="singleSelect"
description=""
            displayName="Chromosome name" name="chr_name">
        <value isSelected="false" displayName="1" name="1"/>
        ...
    </filter>
    ...
</filters>
```

Sample JSON Response

```
[
    {
        "name": "chr_name",
        "displayName": "Chromosome name",
        "description": "",
        "type": "singleSelect",
        "isHidden": false,
```

```
        "qualifier": "",
        "filters": [ ],
        "values": [
            {
                "isSelected": false,
                "name": "15",
                "displayName": "15"
            },
            ...
        ]
    },
    ...
]
```

12.3.3.5. Attributes

Lists all the attributes for the given Dataset(s).

URL
/martservice/attributes

Method
GET

Parameters
datasets (required) - Comma-separated string of Datasets.

config (optional) - The name of the config as returned by the Mart object. You should provide this when the Mart's config is non-empty.

container (optional) - The name of the container you want to return attributes for.

Returns
List of attribute objects

Sample URL
/martservice/attributes?datasets=btaurus_snp&config=snp_config

Sample XML Response

```
<attributes>
    <attribute isHidden="false"
linkURL="/*species2*/Variation/Summary?v=%s%"
            value="" description="" displayName="Variation ID"
name="refsnp_id"/>
    ...
</attributes>
```

Sample JSON Response

```
[
    {
        "name": "refsnp_id",
        "displayName": "Variation ID",
        "description": "",
        "isHidden": false,
        "linkURL": "/*species2*/Variation/Summary?v=%s%",
        "value": "",
        "attributes": [ ]
    },
    ...
]
```

12.3.3.6. Containers

Lists all Containers for the given Mart, starting from the root Container.

Note:
containers and GUI containers are different.

URL
/martservice/containers

Method
GET

Parameters
datasets (required) - Comma-separated string of datasets.

config (optional) - The name of the config as returned by the Mart object. You should provide this when the Mart's config is non-empty.

withattributes (optional) - If true, containers with Attributes (or descendent Containers with Attributes) will be included. Default is true.

withfilters (optional) - If true, Containers with Filters (or descendent Containers with Filters) will be included. Default is true.

Returns
List of Container objects

Sample URL
/martservice/containers?datasets=btaurus_snp&config=snp_config

Sample XML Response

```
<container independent="false" maxAttributes="0"
        maxContainers="0"
        description="root" displayName="root" name="root">
    <containers>
        ...
        <container independent="false" maxAttributes="0"
                maxContainers="0"
                description="NULL"
                displayName="Variation Information"
                name="snpsatts">
            <attribute isHidden="false"
                    linkURL="/*species2*/Variation/Summary?v=%s%"
                    value=""
                    description="" displayName="Variation ID"
                    name="refsnp_id">
                <attributes/>
            </attribute>
            ...
        </container>
        ...
    </containers>
</container>
```

Sample JSON Response

```
{
    "name": "root",
    ...
    "containers": [
        ...
        {
            "name": "snpsatts",
            "displayName": "Variation Information",
            "description": "NULL"
            "maxContainers": 0,
            "maxAttributes": 0,
            "independent": false,
            "attributes": [
                {
                    "name": "refsnp_id",
                    "displayName": "Variation ID",
                    "description": "",
                    "isHidden": false,
                    "linkURL":
"/*species2*/Variation/Summary?v=%s%",
                    "value": "",
                    "attributes": [ ]
                }
                ...
            ],
            "filters": [ ],
            "containers": [ ]
        },
        ...
```

```
    ]
}
```

12.3.3.7. Linkable Datasets

Lists all the Datasets that can be linked with the selected Dataset(s).

URL
/martservice/linkables

Method
GET

Parameters
datasets (required) - Comma-separated string of Datasets.

Returns
List of Dataset objects

Sample URL
/martservice/linkables?datasets=btaurus_snp

Sample XML Response

```
<datasets>
    <dataset isHidden="false" description="hsapiens_snp_som"
            displayName="Homo sapiens Somatic Variation (COSMIC
50)"
            name="hsapiens_snp_som"/>
    ...
</datasets>
```

Sample JSON Response

```
[
    {
        "isHidden": false,
        "name": "hsapiens_snp_som",
        "displayName": "Homo sapiens Somatic Variation (COSMIC
50)",
        "description": "hsapiens_snp_som"
    },
    ...
]
```

12.3.4. Querying

To retrieve data from the REST API, you will need to create a Query XML.

URL
/martservice/results

Method
GET or POST

Parameters
query (required) - The Query XML

download (optional) - If true, then the HTTP response will include the header Content-Disposition: attachment; filename=results.txt Default is false

Returns
The result returned from the Query XML.

12.3.5. Simple Example

The following is an introduction to how the REST API may be used to retrieve data. We will use the curl program in command-line. It is assumed you are accessing the server at http://localhost:9000/. If this is not the case, please change the corresponding URLs.

Retrieve list of Marts.

```
curl http://localhost:9000/martservice/marts
```

Remember one of the Mart's name and config (e.g. name = hsapiens_gene_vega_config, config = hsapians_gene_vega_config)

Retrieve list of Datasets from the Mart.

```
curl
http://localhost:9000/martservice/datasets?mart=hsapiens_gene_veg
a_config
```

Remember one of the Dataset's name (e.g. hsapiens_gene_vega)

Retrieve list of Attributes from the Datasets, and config.

```
curl
http://localhost:9000/martservice/attributes?datasets=hsapiens_ge
ne_vega\
&config=hsapiens_gene_vega_config
```

Remember one of the Attribute's name (e.g. vega_gene_id)

Build your Query XML, and save it to a file query.txt

```
<!DOCTYPE Query>
<Query client="biomartclient" processor="TSV" limit="-1"
header="1">
<Dataset name="hsapiens_gene_vega"
config="hsapiens_gene_vega_config">
    <Attribute name="vega_gene_id"/>
</Dataset>
</Query>
```

Call the results method to get the data.

```
curl --data-urlencode query@query.txt
http://localhost:9000/martservice/results
```

You should see the results printed on your terminal.

12.4 SOAP API

The SOAP API is built on top of the Java API for XML Web Services (JAX-WS). To access the SOAP API you will need tools to help you generate the BioMart SOAP client in the programming language of your choice.

The WSDL can be accessed at the URL /martsoap?wsdl.

In this section we will assume the SOAP service is deployed to http://localhost:9000/martsoap, with the corresponding WSDL file at http://localhost:9000/martsoap?wsdl. If this is not the case, please change the URLs accordingly.

12.4.1. Accessing SOAP API Using Java

Java 5 and newer comes installed with a tool called wsimport, which will build all the client classes needed to interact with a SOAP API.

```
mkdir biomartclient && cd biomartclient
wsimport http://localhost:9000/martsoap?wsdl
```

This will create the *.class files and their packages under the biomartclient directory. You can now use the generated classes to access the SOAP server.

Now create the BioMartSoapClient class that performs some data access. (under the biomartclient directory)

```
// BioMartSoapClient.java

import org.biomart.api.soap.BioMartSoapService;
import org.biomart.api.soap.PortalServiceImpl;
import org.biomart.api.soap.Mart;
import org.biomart.api.soap.Dataset;
import org.biomart.api.soap.Attribute;

public class BioMartSoapClient {
    public static void main(String[] args) {
        BioMartSoapService service = new BioMartSoapService();
        PortalServiceImpl port =
service.getPortalServiceImplPort();
        Mart mart = port.getMarts(null).get(0);
        System.out.println(String.format("Using mart \"%s\"\n",
mart.getDisplayName()));

        Dataset ds = port.getDatasets(mart.getName()).get(0);
        System.out.println(String.format("Using dataset
\"%s\"\n", ds.getDisplayName()));

        Attribute attr = port.getAttributes(ds.getName(), null,
null).get(0);
        System.out.println(String.format("Using attribute
\"%s\"\n", attr.getDisplayName()));

        String query = String.format("<!DOCTYPE Query>"
            + "<Query client=\"javaclient\" processor=\"TSV\"
limit=\"10\" header=\"1\">"
            + "  <Dataset name=\"%s\" config=\"%s\">"
            + "<Attribute name=\"%s\" />"
            + "</Dataset>"
            + "</Query>", ds.getName(), mart.getConfig(),
attr.getName());

        System.out.println("Query is: " + query);

        String results = port.getResults(query);

        System.out.println("Results are:\n" + results);
    }
}
```

Finally, compile and execute the BioMartSoapClient class.

```
javac BioMartSoapClient.java
```

```
java BioMartSoapClient
```

You should see some data output from the Java class.

12.4.2. Accessing SOAP API Using Python

In Python you can use the suds package (easy_install suds) which will read the WSDL and allow you to interact with the SOAP API.

Sample Python Client (biomartclient.py)

```
"""
    biomartclient.py
"""

from suds.client import Client

client = Client("http://localhost:9000/martsoap?wsdl")

mart = client.service.getMarts(None)[0]
print "Using mart '%s'\n" % mart._displayName

ds = client.service.getDatasets(mart._name)[0]
print "Using dataset '%s'\n" % ds._displayName

attr = client.service.getAttributes(ds._name, None, None)[0]
print "Using attribute '%s'\n" % attr._displayName

query = """
<!DOCTYPE Query>
<Query client=\"pythonclient\" processor=\"TSV\" limit=\"10\"
header=\"1\">
<Dataset name=\"%s\" config=\"%s\">
<Attribute name=\"%s\" />
</Dataset>
</Query>""" % (ds._name, mart._name, attr._name)

print "Query is: " + query

results = client.service.getResults(query)

print "Results are:\n" + results
```

Execute the Python script to see the output.

```
python biomartclient.py
```

12.5 Semantic Web API

The Semantic Web API is available through a REST interface. Interactions with the API are carried out through access points as they are defined in MartConfigurator. The access point that is addressed is always part of the URL when using the REST interface.

Two operations are supported by the Semantic Web API:

1. Retrieval of meta-data
2. Querying of data

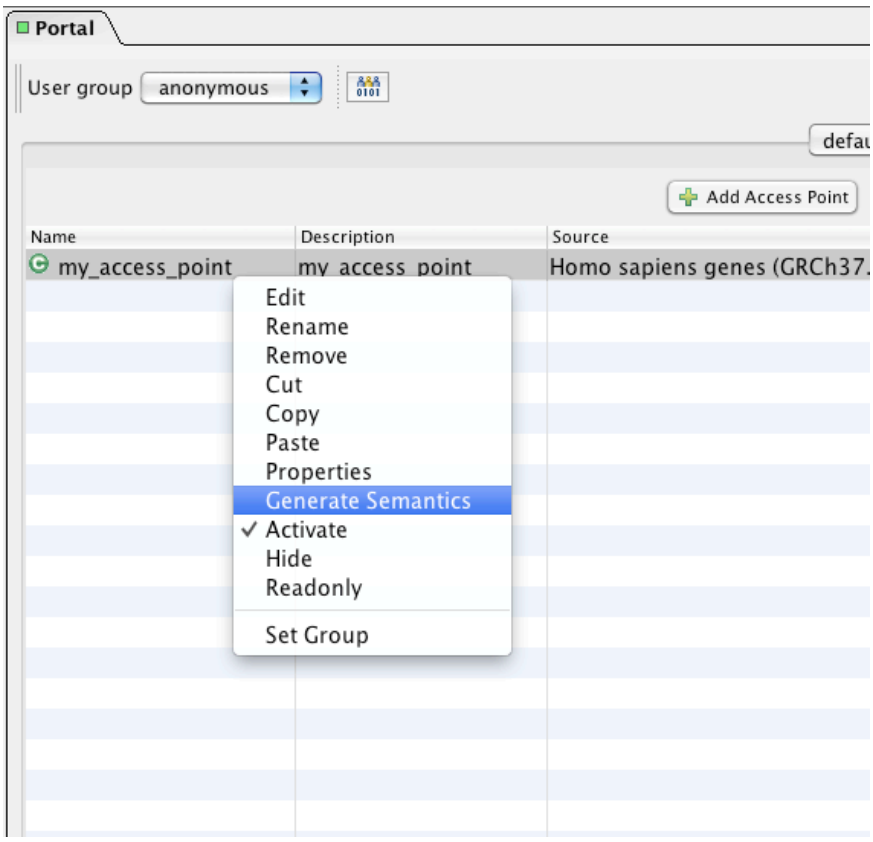
Meta-data of a mart is available on an access-point basis, which allows the retrieval of the datasets, attributes and filters that are linked to the given access point. The meta-data is presented in form of an ontology. Ontologies can be automatically generated using MartConfigurator and they can be viewed using OWL-Tools as described in 12.5.1.

Data queries are carried out using SPARQL. SPARQL queries can be constructed using the web-interface and then submitted via the REST interface to retrieve the results in SPARQL/XML format as described in 12.5.2.

12.5.1. Meta-Data Retrieval: Working with Ontologies

Ontologies are generated within MartConfigurator. For new access points, an ontology is automatically generated and no manual interactions have to be carried out in MartConfigurator. For existing access points an ontology can be generated retrospectively.

Generating an ontology for an existing access point:



All attributes and filters are now part of the ontology as OWL-DatatypeProperties. They are part of the OWL-class that is named after the access point itself: "my_access_point" in this example.

The OWL-ontology can be accessed in a deployed mart under the URL:

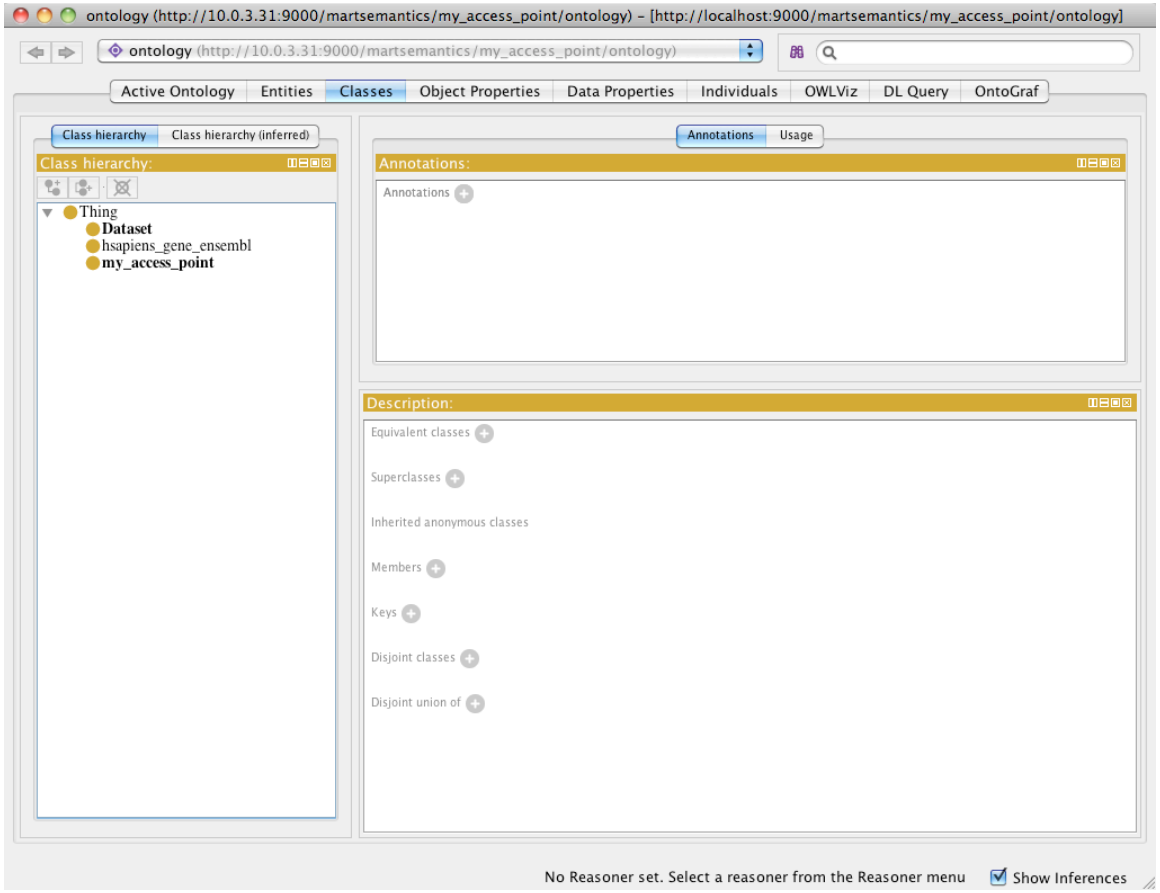
http://*host*/martsemantics/*accesspointname*/ontology

For the access point given above, the URL is:

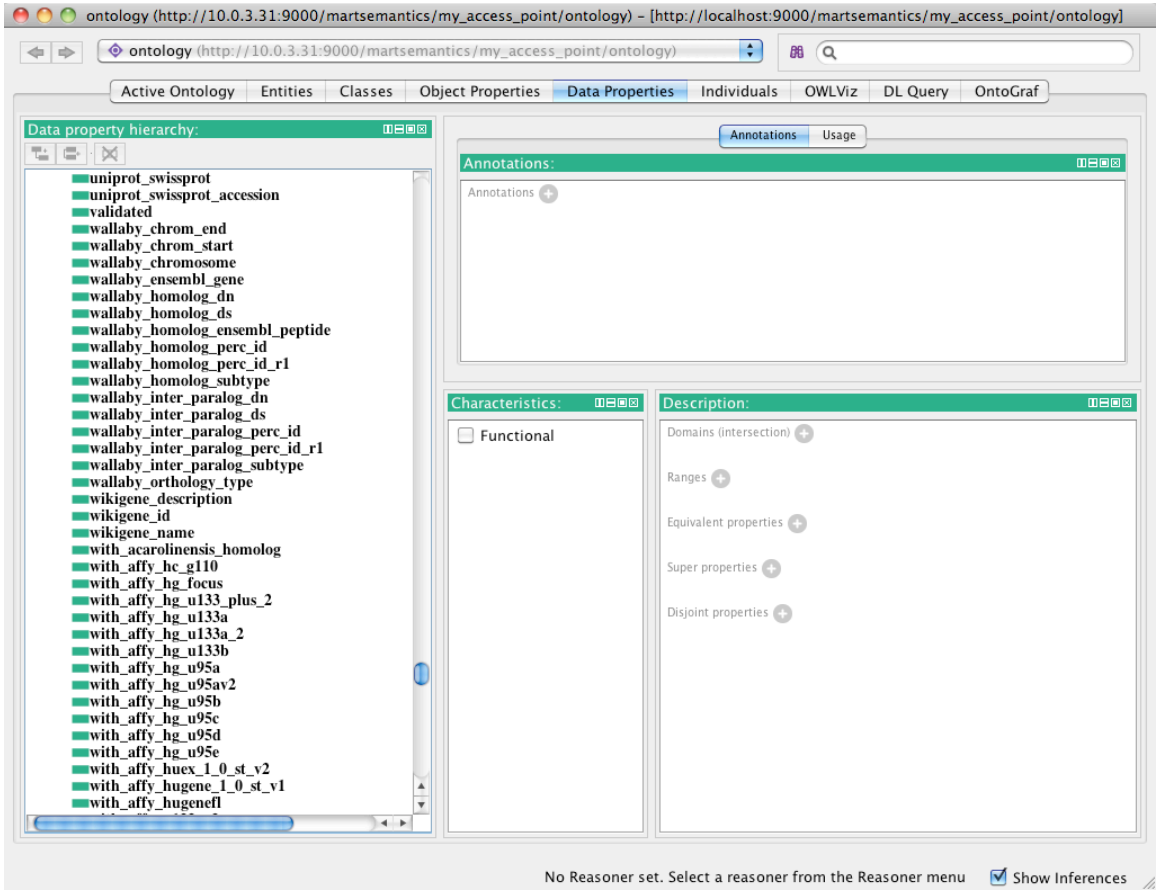
http:://localhost:9000/martsemantics/my_access_point/ontology

The ontology can then be inspected using prevalent OWL-tools such as Protege (http://protege.stanford.edu/).

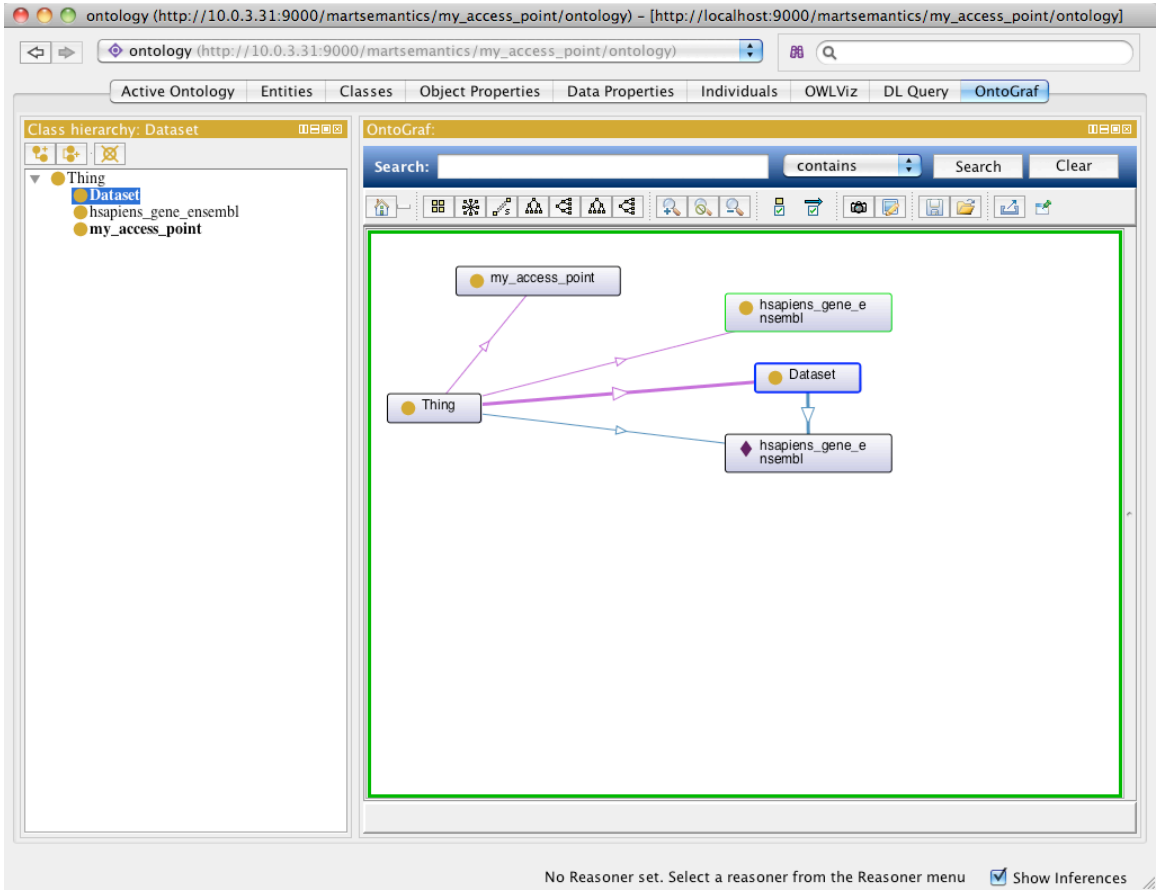
Class-View in Protege:



Property-View in Protege:



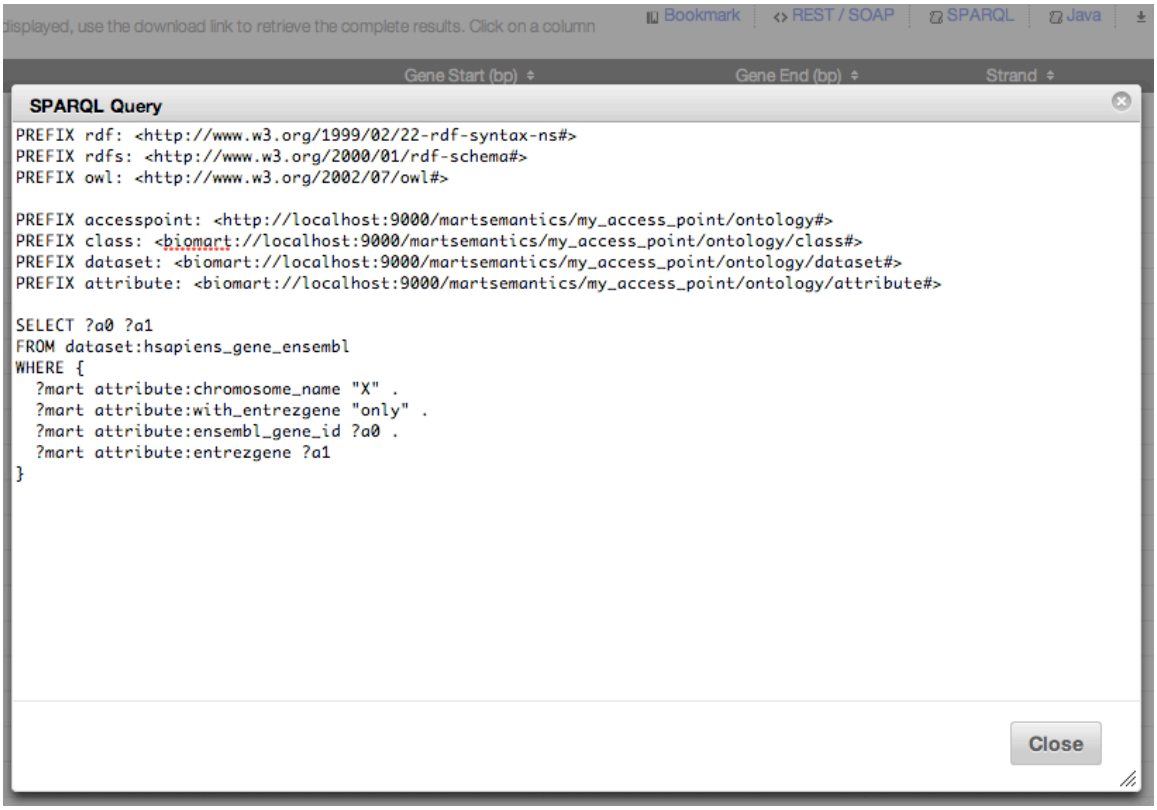
Class-Dependencies in Protege (Showing Dataset Usage):



12.5.2. Semantic Querying: Using SPARQL

SPARQL queries can be generated using the MartView interface. A SPARQL button is shown above the tabular result set for generating a SPARQL query that will return the same result set as shown.

Working with the generated ontology from above, the SPARQL button will for example show:



This query can be submitted via HTTP-GET request. For example, `curl` can be used to make SPARQL queries as follows:

```
curl -H "Accept: application/sparql-results+xml" -v
http://localhost:9000/martsemantics/my_access_point/SPARQLXML/get
/?query=http-encoded-SPARQL-query
```

The result will be returned as SPARQL/XML. For the example SPARQL-query, the result looks as follows:

```
<?xml version="1.0"?>
<!-- BioMart XML-Query:
<Query client="webbrowser" processor="SPARQLXML" limit="-1" header="0">
	<Dataset name="hsapiens_gene_ensembl" config="my_access_point">
		<Filter name="chromosome_name" value="X"/>
		<Filter name="with_entrezgene" value="only"/>
		<Attribute name="ensembl_gene_id"/>
		<Attribute name="entrezgene"/>
	</Dataset>
</Query>
-->
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
	<head>
		<variable name="a0"/>
		<variable name="a1"/>
	</head>
	<results>
		<result>
			<binding name="a0">
				<literal>ENSG00000102128</literal>
```

```
                </binding>
                <binding name="a1">
                        <literal>282808</literal>
                </binding>
        </result>
        <result>
                <binding name="a0">
                        <literal>ENSG00000133169</literal>
                </binding>
                <binding name="a1">
                        <literal>55859</literal>
                </binding>
        </result>
```

(and so on…)

```
        </results>
</sparql>
```

12.6 Java API

The Java API provides public methods to communicate with the local registry. To use the Java API you must first generate a local XML registry file to load from.

12.6.1. MartRegistryFactory

The org.biomart.api.factory.MartRegistryFactory interface declares the method getRegistry, which will return the org.biomart.api.lite.MartRegistry object that will be needed in order to interact with the local registry.

The only implementation currently available is org.biomart.api.factory.XmlMartRegistryFactory, which reads the registry in XML format. You do not need to interact with the MartRegistryFactory instance directly, as you will see later on.

12.6.1.1. Portal

The org.biomart.api.Portal object takes in an instance of MartRegistryFactory and provides public methods for access registry information.

The following are the methods available from the Portal object.

Note:

Parameters annotated with @Nullable is an optional parameter, and should be passed as null if not used. Parameters annotated with @Nonnull are required and must not be null or an empty string (“”).

12.6.1.2. Root GUI Container

Returns the root GUI Container.

Method
getRootGuiContainer

Parameters
@Nullable String type
Restricts returning GUI Containers to this type (e.g. martform)

Returns
org.biomart.api.lite.GuiContainer

12.6.1.3. Marts

Returns a list of Marts.

Method
getMarts

Parameters
@Nullable String guiContainerName
Restricts the returned Marts to the specified GUI Container

Returns
List<org.biomart.api.lite.Mart>

12.6.1.4. Datasets

Returns a list of Datasets.

Method
getDatasets

Parameters
@Nullable String mart
The name of the Mart to return datasets from

Returns

List<org.biomart.api.lite.Dataset>

12.6.1.5. Filters

Returns a list of Filters.

Method
getFilters

Parameters
@Nonnull String datasets
Comma-separated string of Datasets.

@Nullable String config
The name of the config as returned by the Mart object. You should provide this when the Mart's config is non-empty.

@Nullable String container
The name of the container you want to return filters for.

Returns
List<org.biomart.api.lite.Filter>

12.6.1.6. Attributes

Returns a list of Attributes.

Method
getAttributes

Parameters
@Nonnull String datasets
Comma-separated string of Datasets.

@Nullable String config
The name of the config as returned by the Mart object. You should provide this when the Mart's config is non-empty.

@Nullable String container
The name of the container you want to return attributes for.

Returns
List<org.biomart.api.lite.Attribute>

12.6.1.7. Containers

Returns the root Container, which containers all subsequent Containers.

Note:
Containers and GUI Containers are different objects.

Method
getContainers

Parameters
@Nonnull String datasets
Comma-separated string of datasets.

@Nullable String config
The name of the config as returned by the Mart object. You should provide this when the Mart's config is non-empty.

Boolean withattributes
If true, containers with Attributes (or descendent Containers with Attributes) will be included.

Boolean withfilters
If true, Containers with Filters (or descendent Containers with Filters) will be included.

Returns
org.biomart.api.lite.Container

12.6.1.8. Linkables Datasets

Returns a lists of Datasets that can be linked with the Dataset(s) specified.

Method
getLinkables

Parameters
@Nonnull String datasets
Comma-separated string of datasets.

Returns
List<org.biomart.api.lite.Dataset>

12.6.2. Query

The org.biomart.api.Query object provides methods to build and execute a query.

Note:
The Query object's methods return the object itself. This allows for method chaining.

12.6.2.1. Set Processor

Method
setProcessor

Parameters
@Nonnull String processor
Sets the processor that the query engine will run through. Default is TSV.

Returns
org.biomart.api.Query

12.6.2.2. Set Limit

Method
setLimit

Parameters
int limit
The number of result rows to be printed.

Returns
org.biomart.api.Query

12.6.2.3. Enable Header

Method
setHeader

Parameters
boolean header
If true, the results will include a header as the first row. Default is true.

Returns
org.biomart.api.Query

12.6.2.4. Set Client Name

Method
setClient

Parameters
@Nonnull String client
The name of the client executing the query. You can choose a name that includes your organization. Default is biomartclient.

Returns
org.biomart.api.Query

12.6.2.5. Add Dataset

This method returns a org.biomart.api.Query.Dataset object that will allow you to add Filters and Attributes.

Method
addDataset

Parameters
@Nonnull String name
The name of the Dataset.

@Nullable String config
The config of the Mart. Please specify this if it is non-empty.

Returns
org.biomart.api.Query.Dataset

12.6.3. Query.Dataset

The org.biomart.api.Query.Dataset object contains the Filters and Attributes in a Query.

Note:
The Dataset.Query object's methods return the object itself. This allows for method chaining.

12.6.3.1. Add Filter

Method

addFilter

Parameters
@Nonnull String name
The name of the Filter.

@Nonnull String value
The value of the Filter.

Returns
org.biomart.api.Query.Dataset

12.6.3.2. Add Attribute

Method
addAttribute

Parameters
@Nonnull String name
The name of the Attribute.

Returns
org.biomart.api.Query.Dataset

12.6.3.3. Return Query

This method return the original Query object that created the Dataset.

Method
end

Parameters
none

Returns
org.biomart.api.Query

12.6.4. Simple Example

To use the example below, make sure the biomart-0.8.jar file and all other dependencies are on the classpath.

12.6.4.1. Portal Access

The following is a runnable Java class that demonstrates how interaction with the local registry can be done.

```
// PortalExample.java

import java.util.List;
import org.biomart.api.Portal;
import org.biomart.api.factory.MartRegistryFactory;
import org.biomart.api.factory.XmlMartRegistryFactory;
import org.biomart.api.lite.Attribute;
import org.biomart.api.lite.Dataset;
import org.biomart.api.lite.Filter;
import org.biomart.api.lite.Mart;

public class PortalExample {
    public static void main(String[] args) {
        MartRegistryFactory factory = new
XmlMartRegistryFactory("/Users/jhsu/main.xml");
        Portal portal = new Portal(factory);

        List<Mart> marts = portal.getMarts(null);
        System.out.println(String.format("Got %s marts",
marts.size()));

        List<Dataset> datasets =
portal.getDatasets(marts.get(0).getName());
        System.out.println(String.format("Got %s datasets",
datasets.size()));

        List<Filter> filters =
portal.getFilters(datasets.get(0).getName(),
                marts.get(0).getConfigName(), null);
        System.out.println(String.format("Got %s filters",
filters.size()));

        List<Attribute> attributes =
portal.getAttributes(datasets.get(0).getName(),
                marts.get(0).getConfigName(), null);
        System.out.println(String.format("Got %s attributes",
attributes.size()));

        System.exit(0);
    }
}
```

Compile the code and run it.

```
javac -cp [...] PortalExample.java
java -cp [...] PortalExample
```

12.6.4.2. Querying

The following is a runnable Java class that demonstrates how query execution can be done.

```
// QueryExample.java

import org.biomart.api.Portal;
import org.biomart.api.Query;
import org.biomart.api.factory.MartRegistryFactory;
import org.biomart.api.factory.XmlMartRegistryFactory;

public class QueryExample {
    public static void main(String[] args) {
        MartRegistryFactory factory = new
XmlMartRegistryFactory("/path/to/xml");
        Portal portal = new Portal(factory);
        Query query = new Query(portal)
                .setProcessor("TSV")
                .setClient("test")
                .setHeader(true)
                .setLimit(10)
                .addDataset("hsapiens_gene_ensembl",
"hsapiens_gene_ensembl_config")
                    .addFilter("chromosome_name", "1")
                    .addAttribute("ensembl_gene_id")
                .end();
        query.getResults(System.out);
        System.exit(0);
    }
}
```

Compile the code and run it.

```
javac -cp [...] QueryExample.java
java -cp [...] QueryExample
```

13. Troubleshooting Installation

13.1 System checks

This section will help you determine whether the environment requirements have been met before you continue with the rest of the guide.

If the system requirements are not met, you may need to speak to your system administrator to perform an upgrade.

13.1.1. *Nix/OS X users

Open up a Terminal window. Type in the commands below, following greater-than character (>).

1. Check the Java version. BioMart requires Java 6 (1.6.0).

~ > if ["$JAVA_HOME"] ; then $JAVA_HOME/bin/java -version ; else java -version ; fi

```
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04-307-10M3261)
Java HotSpot(TM) 64-Bit Server VM (build 17.1-b03-307, mixed
mode)
```

The Java version printed should be 1.6.x, if you see 1.4.x or 1.5.x then please update your system to use Java 6.

2. Check the ant version. BioMart requires Apache Ant 1.7 or higher.

~ > ant -version

```
Apache Ant version 1.7.1 compiled on June 27 2008
```

If you do not have Ant installed on your system you will see a `command not found` error.

To install the latest version of Ant, please visit the website: http://ant.apache.org/.

13.1.2. Windows users

Open up the Command Prompt and type in the commands below, following the great-than character (>).

1. Check for the correct Java version. BioMart requires Java 6 (1.6.x).

First, check if the JAVA_HOME environment variable is set.

C:\> echo %JAVA_HOME%

C:\Program Files\Java\jdk1.6.0_22

If you see a path printed out, please follow (a), otherwise follow (b).

(a) Check the Java version specified in by JAVA_HOME.

C:\> "%JAVA_HOME%"\bin\java -version

```
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04-307-10M3261)
Java HotSpot(TM) 64-Bit Server VM (build 17.1-b03-307, mixed
mode)
```

(b) Check the Java version of the `java` command.

C:\> java -version

```
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04-307-10M3261)
Java HotSpot(TM) 64-Bit Server VM (build 17.1-b03-307, mixed
mode)
```

The Java version printed from steps (a) or (b) should be 1.6.x, if you see 1.4.x or 1.5.x then please update your system to use Java 6.

2. Check the ant version number. BioMart requires Apache Ant 1.7 or higher.

C:\> ant -version

```
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

If you do not have Ant installed on your system you will see an error message stating that 'ant' is not a program.

To install the latest version of Ant, please visit the website: http://ant.apache.org/.

13.2 Testing Environment

This BioMart 0.8 release candidate is known to work with the following software environments:

Operating systems:

Mac OS X (Leopard and Snow Leopard), Linux (Debian 4.1.1-21)

Web browsers:

Firefox 3.6+, Microsoft IE (7+), Google Chrome (8.0+), Safari 5.0+