InterMine Documentation

Release

InterMine

June 04, 2020

Contents

InterMine is an open source data warehouse build specifically for the integration and analysis of complex biological data.

Developed by the Micklem lab at the University of Cambridge, InterMine enables the creation of biological databases accessed by sophisticated web query tools. Parsers are provided for integrating data from many common biological data sources and formats, and there is a framework for adding your own data. InterMine includes an attractive, user-friendly web interface that works ‘out of the box’ and can be easily customised for your specific needs, as well as a powerful, scriptable web-service API to allow programmatic access to your data.

CHAPTER 1

Contents

1.1 System Requirements

1.1.1 Hardware

Recommendations

The requirements for running InterMine depend on the size of data warehouse you need to create. It is possible to build small InterMine databases on most Linux or Mac desktops but with more substantial databases a more powerful dedicated server is required. The recommendations below are the minimum for running substantial servers such as FlyMine or InterMines for the major model organism databases.

Database servers

The hardware used for a data loading has a **significant** impact on data loading performance. The main recommendations we have are:

- Install plenty of RAM, 16GB or more, but watch out for multiple RAM modules slowing down your RAM access speed.
- Have at least two real CPUs - hyperthreading doesn't count. Preferably have at least four CPUs.
- It is more important to have fast individual CPUs than a lot of CPUs for a build server. InterMine does use multiple threads during data loading, but not asymmetrically - there is one thread which takes a lot of the CPU time. On the other hand, for a production server, having a few more CPUs is more important.
- Have a decent IO subsystem. We currently use a fibrechannel attached RAID array of 16 15krpm discs for our build servers.

Suggestion for a large InterMine instance

- 8 cores
- 32 GB RAM
- ~2TB usable storage (SAS disks are faster than SATA)
  - RAID 10 (4TB raw in RAID 10)
  - hardware RAID controller with a battery backed cache (gives faster write speeds)

  - it doesn't matter whether storage is in the same box or a separate disk array, if separate needs a Fibre Channel connection

- Linux/Unix capable of running Java and PostgreSQL

Note: It's essential to have separate development and production machines.

OS

- Any distribution of Linux/Unix should be fine as long as it runs Java and Postgres, Debian is our preference.
- Use something mainstream and reliable like Linux or BSD
- Use the system that your friendly sysadmin is most familiar with.
- Not favourites:
- Tru64
- *Solaris*

What we use

FlyMine has separate build and production build servers and separate build and production build web servers.

Build

This runs the Java data integration code to build the warehouse, reading from source files/DBs and loading into an intermediate postgres database then the final postgres database. This is write intensive; the faster the disks the better, it only needs 4 cores but the more RAM the better.

Production

This runs the production postgres database. More cores and more RAM means better handling of concurrent requests and more of the database in cache. InterMine often fires a lot of queries at a time for a single user - i.e. when running templates for a report page.

Web server

FlyMine has a separate machine to run Tomcat to serve the webapp, this is the machine that actually runs the live InterMine code. For us this a 4 core machine with 8GB RAM. The cores are more important than the speed, disk space not important, more RAM means better caching.

modENCODE - identical machines

For modENCODE we actually have two identical servers that switch roles with each release. With higher data volumes and more frequent releases this makes more sense as we avoid dumping and reloading. Unlike FlyMine, modMine's database and webapp live on the same server.

Database sizes/ disk space

Disk space on the build and production machines obviously depends on volume of data.

Multiply the database size by at least 3 for a corresponding InterMine instance. This takes into account the various redundant ways we store data and precomputed tables, all to boost query performance.

Precomputed tables are pre-joined tables that can be swapped in dynamically to reduce table joins in actual queries and improve performance. This means a lot of duplicated data is stored.

As a rough guide the current FlyBase database with all 12 genomes is 33GB, an InterMine with this and a couple of extra data sources is 100GB. A full FlyMine release is typically around 500GB.

When running an InterMine build with multiple data sources, database copies are made periodically for backups so there needs to be extra space available, at least four times the final database size.

Related topics:

Solaris Installation guide

Installation notes

Update postgres.conf

autovacuum is not turned off (it's on by default)

Improvements for COPY

```
wal_sync_method = fsync
wal_buffers = 128
checkpoint_segments = 128
bgwriter_percent = 0
bgwriter_maxpages = 0
```

And also for */etc/system* on Solaris 10, 9 SPARC use the following

```
set maxphys=1048576
set md:md_maxphys=1048576
set segmap_percent=50
set ufs:freebehind=0
set msgsys:msginfo_msgmni = 3584
set semsys:seminfo_semmni = 4096
set shmsys:shminfo_shmmax = 15392386252
set shmsys:shminfo_shmmni = 4096
```

Run analyse

Try using the *-fast* compile flag. The binaries might not be portable to other Solaris systems, and you might need to compile everything that links to PostgreSQL with *-fast*, but PostgreSQL will run significantly faster, 50% faster on some tests.

1.1.2 Software

InterMine makes use of a variety of freely available software packages.

Software	At least	Purpose
Git	1.7	check out and update source code
Java SDK	8	build and use InterMine
Tomcat	8.5.x	website
PostgreSQL	9.3.x	database
Perl	5.8.8	run build scripts
Maven	3.0.5	manage local dependencies
SOLR	7.2.1	search engine

Note: InterMine only supports installations onto Linux and Mac OS X systems. Windows systems of any kind are not supported. We run a mixture of Debian and Fedora servers in our data centre in Cambridge.

After installation, most programs require extra configuration to work with InterMine:

Git

Git is our source control software. Download and install git on your local machine.

Note: InterMine is available via JCenter as executable JARs. We do not recommend downloading the InterMine source code.

InterMine source code is available via GitHub.

Getting started

See *Quick Start* or *Create Your Own InterMine!* for instructions on how to create a new InterMine.

Local Installation (for advanced users)

You should use the JARs available via JCenter. However, if you want to make custom changes to InterMine, you can install locally.

1. Get InterMine code

```
~/git $ git clone https://github.com/intermine/intermine.git
```

2. Checkout the InterMine version you need

Get the list of valid tags.

```
# change into the correct directory
~/git $ cd intermine
# get a list of tags
~/git/intermine $ git tag -l
```

Checkout the correct tag for the InterMine version you want to use.

```
# get the correct version of the InterMine software
~/git/intermine $ git checkout tags/<tag_name> -b <branch_name>
```

3. Copy in your changes to the InterMine code.

4. Rebuild JARs locally.

Run the Maven task *install* to compile and create the JARs you need to run an InterMine instance.

```
~/git/intermine $ (cd plugin && ./gradlew clean && ./gradlew install) && (cd intermine && ./gradlew c
```

This places the JARs in *~/.m2/repository*. You can now build a database and deploy a webapp, and your custom local JARs will be used.

Why will Maven use my JARs instead of the published JARs? The Gradle build files are configured so that Maven looks in your local Maven (*~/.m2/respository*) directory first before looking in JCenter. If Maven finds the correct version locally, those are the JARs it will use. But make sure you have the correct version!

Set your InterMine version The InterMine version you use is determined by the system variables set in your mine's *gradle.properties* file.

Make sure you have your *InterMine Versioning Policy* set correctly. If you want to use local JARs, it's best to specify the exact version, e.g. *1.2.3*, of your local JARs. Do this in your mine's *gradle.properties* file.

If you use *4.0.+* there's a possiblity a newer version of InterMine is published. The plus sign instructs Maven to get the latest version of InterMine in *any* repository. In which case, Maven would use the newer JARs in JCenter instead of your local JARs.

Java

We recommend you use OpenJDK rather than Sun's JDK. There isn't much difference now between the two, as far as InterMine is concerned, but going forward it's probably safer.

The version of Gradle we are using is compatible with Java 11.

GRADLE_OPTS

InterMine can be rather memory-intensive, so you will probably need to set memory options for Java. To do this, set the environment variable *GRADLE_OPTS* to pass in to Java, by placing the following line in your *~/.bashrc* file:

```
# ~/.bashrc file
    $ export GRADLE_OPTS="-server -Xmx8g -XX:+UseParallelGC -Xms2g -XX:SoftRefLRUPolicyMSPerMB=1 -XX
```

Run *source* to use this value in the current session.

You should change the *-Xmx* and *-Xms* values if you have very little or very much RAM in your computer.

Building a database requires much more memory than running a webapp only.

Perl

Many of the build processes are carried out by Perl programs. You will need Perl installed on your system to build or maintain an InterMine installation. Linux and MacOS systems will have a suitable Perl already installed. Perl is available for Windows, but is not actively supported by InterMine.

You are encouraged to use http://perlbrew.pl to set up your Perl environment, and make use of the modern toolchain, such as https://metacpan.org/pod/cpanm.

At various times you will be requested to install various Perl modules. Here you can find instructions for how to do this using the native CPAN tool which comes with every Perl distribution on Linux and OSX, using Debian/Ubuntu package managers, as well as manual installs:

CPAN

CPAN stands for the Comprehensive Perl Archive Network - and is the software repository for Perl modules. (you can compare it to http://pypi.python.org/pypi, Yum/Apt repositories in Linux, or even Apple's App Store). If you have Perl you have CPAN. (To check type *cpan* in a terminal).

To install modules with CPAN you may first need to set up the installer: in a terminal run

```
$ cpan
```

This will take you to a cpan shell, which will allow you to configure your properties. to review your current configuration type:

```
$ o conf
```

When you first run cpan should run:

```
$ o conf init
```

This will guide you through the set-up procedure. You can run this later change the settings which are set automatically.

To change a setting manually, type:

```
$ o conf [SETTING NAME] "NEW VALUE"
```

eg to make modules installed uninstall previous versions and use sudo to elevate permissions (very good ideas), type:

```
$ o conf mbuild_install_arg "--uninst 1"
$ o conf mbuild_install_build_command "sudo ./Build"
$ o conf make_install_make_command "sudo make"
```

If you change options, remember to save your changes with:

```
$ o conf commit
```

To install modules, type:

```
$ cpan Module::Name Another::Module::Name
```

To force the install for any reason, use the "-f" flag, so type:

```
$ cpan -f Module::Name
```

Don't forget to use sudo in front of the CPAN command if you have not set the sudo option in the CPAN configuration

DEB Packages

Many Perl libraries are packaged for different Linux distributions. Debian/Ubuntu has a great number of these, and in many cases this is a good alternative to the CPAN install.

The procedure is the same as for any other package:

```
$ sudo apt-get install libxml-writer-perl # installs XML::Writer
```

There is a predictable name to package mapping: "::" becomes "-", there will be a "lib" on the front, and a "-perl" on the end, so:

- "*XML::DOM*" becomes "*libxml-dom-perl*"
- "*Moose*" becomes "*libmoose-perl*"

- and so on

These are the modules you need to build a database:

```
$ sudo apt-get install libxml-writer-perl libxml-sax-base-perl libxml-perl libxml-filter-saxt-perl li
```

To search for a package you can type:

```
$ apt-cache search package-name
```

Manually installing InterMine modules

The InterMine Perl modules are available on CPAN, and you are encouraged to download them from there. However, you can install them manually too. First you will need to check-out the source code. (It is recommended you update your version of Module::Build to at least version 0.36, as this will allow you to automate the dependency installation.)

From your check out (or unzipped tar file) go to the directory "*intermine/perl/*"

```
$ cd git/intermine/perl
```

Here there are three "distributions" of modules you may want to install:

- InterMine-Model
- InterMine-Item (depends on InterMine::Model)
- Webservice-InterMine (depends on InterMine::Model)

The installation procedure for these is the same:

```
$ cd [DISTRIBUTION-DIRECTORY]
$ perl Build.PL            # Checks your system
$ sudo ./Build installdeps # If you have Module::Build >= 0.36
$ ./Build test             # tests the modules: optional but HIGHLY recommended
$ sudo ./Build install     # Installs the modules
```

If you do not have Module::Build 0.36 or above, you can install the dependencies using the above methods (CPAN and Packages).

List of Perl Modules to Install

- For the InterMine modules:
- *List::MoreUtils* (utility functions for handling lists)
- *LWP* (Handling network communication)
- *Module::Find* (Automatically locating modules by name)
- *Moose* (Object system)
- *MooseX::Role::WithOverloading* (Allows roles to overload operators)
- *MooseX::Types* (Type constraint system)
- *Text::CSV_XS* (Processing .csv and .tsv files)
- *URI* (Handling urls)
- *XML::Parser::PerlSAX* (Parsing XML)
- *XML::DOM* (XML processing and output)

- *Text::Glob* (used by the *project_build* script)
- for the download scripts:
- Log::Handler
- DateTime
- Module::Find
- Web::Scraper
- Ouch
- Number::Format
- PerlIO::gzip
- Perl6::Junction
- for generating InterMine Items XML:
- Getopt::Std
- Log::Handler;
- Digest::MD5

How to install all the Perl Modules to Run the Data Downloader Script

In order to download all the Perl scripts required by the Data Downloader script, use the following cpan installation command:

```
$ cpan install DateTime Module::Find Web::Scraper Ouch Number::Format PerlIO::gzip Perl6::Junction L
```

PostgreSQL

Installing PostgreSQL

Important: We recommend you install PostgreSQL 9.2 and above. We currently run our continuous integration tests on PostgreSQL 9.2. The PostgreSQL downloads page has packages for most systems that set up everything for you.

Fedora/CentOS http://wiki.openscg.com/index.php/PostgreSQL_RPM_Installation

Debian/Ubuntu *sudo apt-get install postgresql*

Mac There are several good options:

- Postgres.app - Very easy for a development machine, requires zero configuration.
- MacPorts
- Homebrew
- Manually

We have had good experiences with Postgres.app and Macports.

Some of the recommended setting below may not apply to older versions of PostgreSQL.

Configuration file Most of the configurations below are made updating the file *postgresql.conf*, usually located in */etc/postgres/version-nr/main*.

Required Configurations

Allow remote connections

listen_addresses	'*'
port	5432

Recommended Configurations The system works reasonably well with the default configuration. For better performance we recommend to make the changes below.

Character Set Encoding You should only use either *SQL_ASCII* or *UTF-8*. If performance is an issue, the use of *SQL_ASCII* is strongly recommended. [1]

Procedures to change character encoding to *SQL_ASCII* in PostgreSQL 9.x:

```
sudo -u postgres psql
update pg_database set datallowconn = TRUE where datname = 'template0';
\c template0
update pg_database set datistemplate = FALSE where datname = 'template1';
drop database template1;
create database template1 with template = template0 encoding = 'SQL_ASCII' LC_COLLATE='C' LC_CTYPE='
update pg_database set datistemplate = TRUE where datname = 'template1';
\c template1
update pg_database set datallowconn = FALSE where datname = 'template0';
\q
exit
```

you can check the expected screenshot here [2] .

Database Server Configuration

Kernel Memory setting Please check your server kernel setting

```
getconf PAGE_SIZE
getconf _PHYS_PAGES

sysctl -a | grep -E "shmall|shmmax"
```

(use sudo if necessary)

Set

[1] The InterMine system stores all text in the database in *UTF-8* format. If you set PostgreSQL to *LATIN-9*, then PostgreSQL will perform some incorrect conversions, and may even give an error. Setting the format to *UTF-8* results in PostgreSQL treating the text completely correctly, which is quite a complicated and slow operation in *UTF-8*.

If you set PostgreSQL to *SQL_ASCII*, then that is a special character set in Postgres, which basically means "do no conversions". This is sufficient for almost all operations. All comparisons and index lookups will be done on a byte-by-byte basis, which is much faster than having to deal with Unicode's complications.

Please try to treat InterMine as a black box. The fact that it uses PostgreSQL to store its data should be a detail that should be hidden as much as possible. The InterMine system is written in Java, and therefore handles all text in Unicode.

The template1 database is the database used as a template when you run the *createdb* command. Update the encoding for template1 to be SQL_ASCII then every database you create from now on will have the correct encoding.

[2]

```
shmall = phys_pages / 2
shmmax = shmall * pagesize
```

by editing the file

> /etc/sysctl.d/30-postgresql-shm.conf

and sourcing it

> sudo sysctl -p /etc/sysctl.d/30-postgresql-shm.conf

PostgreSQL parameters For better performance. Read http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server for more information.

Parameter	Suggested value (build)
shared_buffers	10-25% of RAM
temp_buffers	around 80MB
work_mem	around 500MB but < 10% of RAM
maintenance_work_mem	5% of RAM but < 20% of RAM
default_statistics_target	around 250
random_page_cost	around 2.0-2.5
effective_cache_size	50% of RAM
synchronous_commit	off
geqo_threshold	14
from_collapse_limit	14
join_collapse_limit	14
max_locks_per_transaction	640
max_pred_locks_per_transaction	640
checkpoint_segments	128
checkpoint_timeout	10min
checkpoint_completion_target	0.9

Note that most of the changes above require starting postgres.

Note: Depending on your system configuration (production or development), the type of sources used in the build (files or databases) and the load on your web application, you may need to increase the **max_connections** parameter (for example to 250).

Client Authentication You should also add a line to the pg_hba.conf file to allow logging in via password:

```
host    all         all         0.0.0.0/0             password
```

```
postgres=# update pg_database set datallowconn = TRUE where datname = 'template0';
UPDATE 1
postgres=# \c template0
You are now connected to database "template0" as user "postgres".
template0=# update pg_database set datistemplate = FALSE where datname = 'template1';
UPDATE 1
template0=# drop database template1;
DROP DATABASE
template0=# create database template1 with template = template0 encoding = 'SQL_ASCII' LC_COLLATE='C
CREATE DATABASE
template0=# update pg_database set datistemplate = TRUE where datname = 'template1';
UPDATE 1
```

```
template0=# \c template1
You are now connected to database "template1" as user "postgres".
template1=# update pg_database set datallowconn = FALSE where datname = 'template0';
UPDATE 1
```

See also: *HikariCP and InterMine settings*

HikariCP and InterMine settings

InterMine now uses HikariCP as its default JDBC connection pool. If this is not available, InterMine will use the default PostgreSQL distribution, *PGPoolingDataSource*.

The maximum number of connections allocated to a database, set in the properties files, is now reserved at the start and it is a ceiling to the number of connections a database can reach. As a consequence, setting for the previously used connection pool could exhaust the PostgreSQL allocation at start up (either of building an InterMine database or a web application).

> Reasonable settings for `datasource.maxConnections` are 20 for the production database and 5 for other databases. You may need to increase your Postgres parameter `max connections`, for example to 250.

Note: The InterMine property `datasource.maxConnections` corresponds to the `maximumPoolSize` of the HikariCP.

Which properties? The *maxConnections* property is set in *default.intermine.production.properties* in InterMine:

```
# in intermine-resources
# default.intermine.production.properties
db.production.datasource.class=com.zaxxer.hikari.HikariDataSource
db.production.datasource.dataSourceClassName=org.postgresql.ds.PGSimpleDataSource
db.production.datasource.dataSourceName=db.production
#db.production.datasource.serverName=server_name
#db.production.datasource.databaseName=db_name
#db.production.datasource.user=user
#db.production.datasource.password=password
db.production.datasource.maxConnections=20
db.production.driver=org.postgresql.Driver
db.production.platform=PostgreSQL
```

You can override this propery in your mine's property file.

Tomcat You'll also need to update your Tomcat settings. Add *clearReferencesStopTimerThreads* to your $TOMCAT/conf/context.xml file, so it should look like so:

```
<Context sessionCookiePath="/" useHttpOnly="false" clearReferencesStopTimerThreads="true">
...
</Context>
```

Other HikariCP configurations While HikariCP default settings are good, there could be situations where some changes could be useful. HikariCP provides a good number of parameters that can be set (see the configuration section at https://github.com/brettwooldridge/HikariCP).

For example, sometime it can be useful, to avoid exceeding the number of connections set in the database, to set the minimumIdle number of connections. This could be the case in development and when deploying multiple webapps.

For performance purposes is nevertheless suggested by Hikari people to have *minimumIdle* = *maximumPoolSize* (InterMine maxConnections). To set a minimumIdle parameter just add a line like the following to your mine's properties file:

```
db.production.datasource.minimumIdle=10
```

Further reading https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing

https://groups.google.com/forum/#!forum/hikari-cp

Tomcat

Installation

Warning: Several people have had problems with Tomcat installations set up by Linux package managers, mainly due to permissions issues. We recommend installing according to the these instructions instead.

The quickest way to get a working Tomcat:

1. Download the latest stable binary distribution *tar.gz* from the Apache Tomcat site.
2. Unpack it:

```
$ tar -zxvf apache-tomcat-x.y.z.tar.gz
```

After Installation

Users Set up a tomcat user with the 'manager' role by editing *conf/tomcat-users.xml*:

```
<tomcat-users>
   <role rolename="manager-gui"/>
   <role rolename="manager-script"/>
   <user username="manager" password="manager" roles="manager-gui,manager-script"/>
</tomcat-users>
```

You can check this works by accessing the manager interface at http://localhost:8080/manager/html

If you used a package manager to get Tomcat, the manager may not be included. Here's the Debian command you need:

```
$ apt-get install tomcat7-admin
```

Context Edit context.xml:

```
<Context sessionCookiePath="/" useHttpOnly="false" clearReferencesStopTimerThreads="true">
...
</Context>
```

Server XML You also need to check in your *server.xml* file that the correct *UTF-8* encoding has been applied to all connectors in use (see CharacterEncoding). Make sure that every connector element in use reads as follows:

```
<Connector ... URIEncoding="UTF-8"/>
      ...
</Connector>
```

Without this, permalinks may break.

startup.sh Add this property in startup.sh:

```
JAVA_OPTS="$JAVA_OPTS -Dorg.apache.el.parser.SKIP_IDENTIFIER_CHECK=true"
export JAVA_OPTS
```

Tomcat 7.0 has improved the enforcement of Expression Language rules and by default doesn't allow the use of Java keywords. This flag makes Tomcat 7 more permissive.

If tomcat is installed as service, add org.apache.el.parser.SKIP_IDENTIFIER_CHECK=true to /etc/tomcat7/catalina.properties instead of adding JAVA_OPTS in startup.sh.

Starting Tomcat If Tomcat isn't already running, start it with this command:

```
# from tomcat/bin
$ ./startup.sh
```

Visit the Tomcat manager at http://localhost:8080/. The username and password required to access the manager are *webapp.manager* and *webapp.password* as specified in your Mine properties file.

Stopping Tomcat To stop Tomcat, run this command:

```
# from tomcat/bin
$ ./shutdown.sh
```

You can't drop a database if Tomcat has an open connection to a Postgres database. You have to:

1. undeploy your webapp
2. restart tomcat
3. dropdb

Common Errors

Out of Memory Errors To avoid *java.lang.OutOfMemory* errors, specify the JVM heap size in *$TOMCAT_HOME/bin/tomcat.sh*. You can specify the size as part of *TOMCAT_OPTS*:

```
'-Xmx256m -Xms128m'
```

Session Errors If you get a "Session Error" when you start up your webapp, you may need to update your Tomcat configuration to remove application path in front of sessions' cookies.

You get this error because the home page makes several requests but your session is lost between transactions with a new session started with the first query. For instance, when you go to the beta.flymine.org home page your cookie path will initially be "/". To display the "most popular" template queries, a query is run on full URL using the path "/beta". The session with the "/" path is discarded and a new session cookie is created with the "/beta" path. (You can view the values stored in your cookies via your web browser.)

Add these 2 attributes to *$TOMCAT/conf/context.xml*

```
sessionCookiePath="/"
useHttpOnly="false"
```

Tomcat 7.0 context documentation

Mac Installation Notes

This is a work in progress, at the moment just some notes and useful links.

Installing Tomcat

Tomcat is easy to install by downloading the tar.gz file of the latest release and extracting it, then you can get started almost immediately.

From the Tomcat website find the latest stable version in the Downloads section, currently 6.0.x. Scroll to 'Binary Distributions', 'Core' and save the *tar.gz* file.

Extract this file with:

```
$ tar -zxvf apache-tomcat-6.0.x
```

Change into the apache-tomcat-6.0.x, the following directories are of interest:

- *bin* - scripts to startup and shutdown tomcat
- *logs* - error logs are written here
- *webapps* - the directory web applications are deployed to
- *conf* - configuration files

Before starting you need to set up a mananger user so you can deploy web applications and we recommend you allocate more RAM to tomcat - [wiki:Prerequisites#Tomcat see here]

Start up tomcat by running:

```
$ apache-tomcat-6.0.x/bin/startup.sh
```

To check tomcat is running try to access *localhost:8080* in a web browser, you should see a Tomcat home page. If you don't see the tomcat page check *apache-tomcat-6.0.x/catalina.out* and *apache-tomcat-6.0.x/localhost-<data>.log* for error messages and consult the Tomcat docs

Installing Eclipse

Eclipse is a free, open-source Java editing environment, configuration to open the various code modules in InterMine as Eclipse projects is included in the checkout. You can download the Mac OS X version of Eclipse from http://www.eclipse.org/downloads, either the standard or EE edition will work fine. Just unzip the download and it will work immediately.

See our guide to setting up InterMine in Eclipse: EclipseSetup.

For convenient startup from the Terminal command line you can put *eclipse* in your *$PATH* or create a link to it, for example:

```
$ sudo ln -s /Applications/eclipse/eclipse /usr/local/bin/eclipse
```

Apple has a guide to Eclipse at http://developer.apple.com/tools/eclipse.html.

Installing Postgres

See http://www.postgresql.org/download/macosx. We've had the most success with MacPorts.

Maven

InterMine uses Maven to manage local dependencies, including your mine-specific data sources.

```
# for Ubuntu
sudo apt-get install maven
```

Previously you had to download and compile InterMine. Now, instead, you'll be using the compiled InterMine JARs available via jCenter. They will be automatically dowloaded and stored in the gradle cache ~/.gradle/caches/modules-2/files-2.1/org.intermine/.

To use your mine-specific bio sources, you will install Maven locally. The install task, recompiles the bio-source code, creates a new jar and installs it in you local Maven. These JARs are located in ~/.m2.

Intellij

It's recommended that if you are working with InterMine's Java code, you use an IDE. Our favourite IDE at InterMine towers is Intellij.

Intellij has a Gradle plugin so it automatically handles Gradle projects correctly. Here are detailed instructions that are quite clear:

https://www.jetbrains.com/help/idea/gradle.html

Depending on your Intellij version you should:

1. New >
2. Project from existing source >
3. Then select the *build.gradle* file from your *bio/sources* directory.

Warning: Don't select the *build.gradle* files from your sources. Instead, select the *build.gradle* file from the *bio/sources* directory. The dependencies are listed in the main project's *build.gradle* file. If you load each subproject by itself, Intellij won't load the dependencies correctly.

You should import two projects for your mine:

- flymine (webapp and dbmodel)
- flymine-bio-sources

If you are working with the core InterMine code for whatever reason, we have several projects to import:

- plugin
- intermine
- bio
- bio-sources
- bio-postprocess

Errors

You'll get errors at first as the dependencies are not in place. Build each project, and the dependencies will be downloaded and put on your classpath. We recommend you check the option "Build project automatically", located under Compiler Settings.

Running Unit Tests

To run a unit test, right click on the "test" task and execute. "Test" is under "Verification".

Here is a detailed explanation:

https://www.jetbrains.com/help/idea/gradle.html#gradle_context_menu

Solr

InterMine uses Solr for its keyword search. Now the first search is instant, you can inspect the search index directly (via http://localhost:8983/solr/) and there's a facet web service (via */service/facet-list* and */service/facets?q=gene*). Certain bugs, e.g. searching for the gene "OR", are also now fixed.

See below for how to install and configure Solr search for your InterMine

Configure the InterMine instance

Configure the search end point

```
# keyword_search.properties
# replace "flymine" with your mine's name
index.solrurl = http://localhost:8983/solr/flymine-search
index.batch.size = 1000
```

Configure the autocomplete

```
# objectstoresummary.config.properties
# replace "flymine" with your mine's name
autocomplete.solrurl = http://localhost:8983/solr/flymine-autocomplete
```

Configure XML library

```
# your mine's gradle.properties
systemProp.javax.xml.stream.XMLOutputFactory = com.sun.xml.internal.stream.XMLOutputFactoryImpl
```

Otherwise the com.ctc.wstx.stax.WstxOutputFactory class is loaded. See #1889 for details.

Install SOLR

Download Solr binary package and extract it to any place you like. Inside */solr-7.2.1* directory start the server with this command:

```
# Starts the server instance on port 8983
solr-7.2.1 $ ./bin/solr start
```

Initialising Search Indexes

Note: Be sure your $GRADLE_OPTS parameter is set correctly so you have enough memory and disk space for the search index.

To create a Intermine collection for search process, run this command inside the solr directory.

```
# Initialises the search index
# replace "flymine-search" with whatever you configured above in the properties file
solr-7.2.1 $ ./bin/solr create -c flymine-search
```

To create a Intermine collection for autocomplete process, run this command inside the solr directory.

```
# Initaliases the autocomplete index
# replace "flymine-autocomplete" with whatever you configured above in the properties file
solr-7.2.1 $ ./bin/solr create -c flymine-autocomplete
```

Create Search Indexes

To populate your search index, you'll need to add postprocesses to your mine's project XML file: *create-search-index* and *create-autocomplete-index*.

See *Project XML* and *Post processing* for details.

Configuring Search Results

See *Keyword Search* for details on how to configure the search results.

Production search

You can easily copy your index from your dev to your production server. You can copy the entire */solr* directory then do *./bin/solr start*. You can also dump / restore the index. Be sure to copy the *managed-schema* file over as well the first time. Don't forget to restart Solr after making changes.

InterMine uses `Gradle` to manage the build but do not install Gradle locally. Instead use the wrapper provided.

1.2 Get started

1.2.1 Tutorial

Following the steps on this page you will set up an example InterMine. You will:

- Load some real data sets for Malaria (*P. falciparum*)
- Learn about how data integration works
- Deploy a webapp to query the data

Note: See *Quick Start* if you are impatient and just want to run an InterMine.

Getting Started

Software

We use git to manage and distribute source code and gradle as our build system. For this tutorial you will need the following software packages installed locally and running:

- PostgreSQL
- Git
- Java
- Tomcat
- Solr
- Perl (for the final build script)

See *Software* for configuration details.

BioTestMine

Download the mine code from GitHub.

```
$ mkdir git
$ cd git
~/git $ git clone https://github.com/intermine/biotestmine
```

Get rid of daemons

Gradle has helper processes enabled by default. We're going to disable those by setting *-Dorg.gradle.daemon=false*

```
~/git $ export GRADLE_OPTS="-Dorg.gradle.daemon=false"
```

Help! Something's gone wrong.

If at any point you need help or have a quick (or not so quick) question, please get in touch! We have a discord server, twitter and a developer mailing list.

BioTest Mine

BioTestMine is a dummy test mine we use to test out new features which contains real (old) data for Malaria (*P. falciparum*).

To get started, change into the directory you checked out the BiotestMine source code to and look at the sub-directories:

```
~/git $ cd biotestmine
~/git/biotestmine $ ls
```

directory/file	purpose
/dbmodel	contains information about the data model and related configuration files
/webapp	basic configuration for the webapp
/data	contains a tar file with data to load
build.gradle	The *–stacktrace* option will display complete error messages if there is a problem.
gradle.properties	Sets system variables. Determines which version of InterMine you use.
settings.gradle	Sets gradle projects. Do not edit.
project.xml	Configures which data parsers are run during your build.

There is also a gradle directory (*/gradle*) and executables (*gradlew, gradle.bat*).

Project.xml

The *project.xml* allows you to configure which data to load into your Mine. The file has two sections: sources and post-processing.

<sources> The <*source*> elements list and configure the data sources to be loaded. A source can have a name and a type.

type Corresponds to the name of the bio-source artifact (jar) which includes parsers to retrieve data and information on how it will be integrated.

name can be anything and can be the same as *type*, using a more specific name allows you to define specific integration keys.

<*source*> elements can have several properties depending on source type: *src.data.dir*, *src.data.file* and *src.data.includes* are all used to define locations of files that the source should load. Other properties are used as parameters to specific parsers.

<post-processing> Specific operations can be performed on the Mine once data is loaded, these are listed here as <*post-process*> elements. We will look at these in more detail later.

Data to load

The biotestmine git repository includes a tar file with data to load into BiotestMine. These are real, complete data sets for *P. falciparum* (but very old!).

We will load genome annotation from PlasmoDB, protein data from UniProt and GO annotation also from PlasmoDB.

See `/get-started/tutorial/test-data` for details on the data.

Copy this to a local directory (your home directory is fine for this workshop) and extract the archive:

```
$ cd
$ cp git/biotestmine/data/malaria-data.tar.gz .
$ tar -zxvf malaria-data.tar.gz
```

A dummy project XML file is available in the */data/* directory. Copy it into your *biotestmine* directory, then edit *project.xml* to point each source at the extracted data, just replace */data* with */home/username* (or on a mac */Users/username*). Do use the absolute path.

```
$ cd ~/git/biotestmine
$ cp ~/git/biotestmine/data/project.xml .
~/git/biotestmine $ sed -i 's/\DATA\_DIR/\/home\/username/g' project.xml
```

For example, the *uniprot-malaria* source:

```
<sources>
  <source name="uniprot-malaria" type="uniprot">
    <property name="uniprot.organisms" value="36329"/>
    <property name="src.data.dir" location="/home/username/malaria/uniprot/"/>
  </source>
  ...
```

Note: All file locations must be absolute not relative paths.

The *project.xml* file is now ready to use.

Properties file

Configuration of local databases and tomcat deployment is kept in a *MINE_NAME.properties* file in a *.intermine* directory under your home directory. We need to set up a *biotestmine.properties* file.

If you don't already have a *.intermine* directory in your home directory, create one now:

```
$ cd
$ mkdir .intermine
```

There is a partially completed properties file for BioTestMine already. Copy it into your *.intermine* directory:

```
$ cd
$ cp git/biotestmine/data/biotestmine.properties .intermine/
```

Update this properties file with your postgres server location, username and password information for the two databases you just created. The rest of the information is needed for the webapp and will be updated later.

For the moment you need to change *PSQL_USER* and *PSQL_PWD* in the *db.production* and *db.common-tgt-items* properties.

```
# Access to the postgres database to build into and access from the webapp
db.production.datasource.serverName=localhost
# port: uncomment the next line if use different port other than 5432
# db.production.datasource.port=PORT_NUMBER
db.production.datasource.databaseName=biotestmine
db.production.datasource.user=PSQL_USER
db.production.datasource.password=PSQL_PWD
```

If you don't have a password for your postgres account you can leave *password* blank.

Create databases

Finally, we need to create *biotestmine* and *items-biotestmine* postgres databases as specified in the *biotest-mine.properties* file:

```
$ createdb biotestmine
$ createdb items-biotestmine
```

New postgres databases default to *UTF-8* as the character encoding. This will work with InterMine but performance is better with *SQL_ASCII*.

The Data Model

Now we're ready to set up a database schema and load some data into our BioTestMine, first some information on how data models are defined in InterMine.

Defining the model

InterMine uses an object-oriented data model, classes in the model and relationships between them are defined in an XML file. Depending on which data types you include you will need different classes and fields in the model, so the model is generated from a core model XML file and any number of *additions* files. These additions files can define extra classes and fields to be added to the model.

- Elements of the model are represented by Java classes and references between them.
- These Java classes map automatically to tables in the database schema.
- The object model is defined as an XML file, that defines *classes*, their *attributes* and *references* between classes.
- The Java classes and database schema are automatically generated from an XML file.

The model is generated from a core model XML file and any number of additions files defined in the db-model/build.gradle file.

core.xml The core InterMine data model is defined in core.xml file.

Note the fields defined for *Protein*:

```
<class name="Protein" extends="BioEntity" is-interface="true">
  <attribute name="md5checksum" type="java.lang.String"/>
  <attribute name="primaryAccession" type="java.lang.String"/>
  <attribute name="length" type="java.lang.Integer"/>
  <attribute name="molecularWeight" type="java.lang.Double"/>
  <reference name="sequence" referenced-type="Sequence"/>
  <collection name="genes" referenced-type="Gene" reverse-reference="proteins"/>
</class>
```

Protein is a subclass of *BioEntity*, defined by *extends="BioEntity"*. The *Protein* class will therefore also inherit all fields of *BioEntity*.

```
<class name="BioEntity" is-interface="true">
  <attribute name="primaryIdentifier" type="java.lang.String"/>
  <attribute name="secondaryIdentifier" type="java.lang.String"/>
...
```

Sequence Ontology

```
mineDBModelConfig {
  modelName = "genomic"
  extraModelsStart = "so_additions.xml genomic_additions.xml"
  extraModelsEnd = ""
}
```

The first file merged into the core model is the *so_additions.xml* file. This XML file is generated in the *dbmodel/build* directory from terms listed in the *so_terms* file, as configured in the dbmodel/build.gradle file.

```
dbModelConfig {
  soTermListFilePath = "dbmodel/resources/so_terms"
  soAdditionFilePath = "dbmodel/build/so_additions.xml"
}
```

The build system creates classes corresponding to the Sequence Ontology terms.

Additions files The model is then combined with any extra classes and fields defined in the sources to integrate, those listed as <*source*> elements in *project.xml*. Look at the additions file for the UniProt source, for example. This defines extra fields for the *Protein* class which will be added to those from the core model.

Creating a database

Now run the gradle task to merge all the model components, generate Java classes and create the database schema:

```
# creates the empty database tables
~/git/biotestmine $ ./gradlew buildDB
```

The clean task is necessary when you have run the task before, it removes the *build* directory and any previously generated models.

This task has done several things:

1. Merged the core model with other model additions and created a new XML file:

```
~/git/biotestmine $ less dbmodel/build/resources/main/genomic_model.xml
```

Look for the *Protein* class, you can see it combines fields from the core model and the UniProt additions file.

2. The *so_additions.xml* file has also been created using the sequence ontology terms in *so_term*:

```
~/git/biotestmine $ less dbmodel/build/so_additions.xml
```

Each term from *so_term* was added to the model, according to the sequence ontology.

3. Generated and compiled a Java class for each of the <*class*> elements in the file. For example *Protein.java*:

```
~/git/biotestmine $ less dbmodel/build/gen/org/intermine/model/bio/Protein.java
```

Each of the fields has appropriate getters and setters generated for it, note that these are *interfaces* and are turned into actual classes dynamically at runtime - this is how the model copes with multiple inheritance.

4. Automatically created database tables in the postgres database specified in *biotestmine.properties* as *db.production* - in our case *biotestmine*. Log into this database and list the tables and the columns in the protein table:

```
$ psql biotestmine
biotestmine=#  \d
biotestmine=#  \d protein
```

The different elements of the model XML file are handled as follows:

attributes there is one column for each attribute of *Protein* - e.g. *primaryIdentifer* and *length*.

references references to other classes are foreign keys to another table - e.g. *Protein* has a reference called *organism* to the *Organism* class so in the database the *protein* table has a column *organismid* which would contain an id that appears in the *organism* table.

collections indirection tables are created for many-to-many collections - e.g. *Protein* has a collection of *Gene* objects so an indirection table called *genesproteins* is created.

This has also created necessary indexes on the tables:

```
biotestmine=#  \d genesproteins
```

Warning: Running *buildDB* will destroy any existing data loaded in the biotestmine database and re-create all the tables.

The model XML file is stored in the database once created, this and some other configuration files are held in the *intermine_metadata* table which has *key* and *value* columns:

```
biotestmine=# select key from intermine_metadata;
```

Loading Data

Now we have the correct data model and the correct empty tables in the database. We can now run several data parsers to load our data into our database.

For this tutorial we will run several data integration and post-processing steps manually. This is a good way to learn how the system works and to test individual stages. For running actual builds there is a *project_build* script that will run all steps specified in *project.xml* automatically. We will cover this later.

Loading data from a source

Loading of data is done by running the *integrate* gradle task.

```
# load the uniprot data source
~/git/biotestmine $ ./gradlew integrate -Psource=uniprot-malaria --stacktrace
```

./gradlew	Use the provided gradle wrapper so that we can be sure everyone is using the same version.
integrate	Gradle task to run the specified data source
-Psource=	Data source to run. Source name should match the value in your project XML file
–stacktrace	The *–stacktrace* option will display complete error messages if there is a problem.

This will take a couple of minutes to complete, the command runs the following steps:

1. Checks that a source with name *uniprot-malaria* exists in *project.xml*
2. Reads the UniProt XML files at the location specified by *src.data.dir* in the *project.xml* file
3. Runs the parser included in the UniProt JAR. The JARs for every core InterMine data source are published in JCenter. The build looks for jar with the name matching “bio-source-<source-type>-<version>.jar”, e.g. *bio-source-uniprot-2.0.0.jar*. Maven will automatically download the correct JARs for you.
4. The UniProt data parser reads the original XML and creates *Items* which are metadata representations of the objects that will be loaded into the biotestmine database. These items are stored in an intermediate *items* database (more about *Items* later).
5. Reads from the *items* database, converts items to objects and loads them into the biotestmine database.

This should complete after a couple of minutes. Now that the data has loaded, log into the database and view the contents of the protein table:

```
$ psql biotestmine
biotestmine#  select count(*) from protein;
```

And see the first few rows of data:

```
biotestmine#  select * from protein limit 5;
```

Object relational mapping

InterMine works with objects, objects are loaded into the production system and queries return lists of objects. These objects are persisted to a relational database. Internal InterMine code (the ObjectStore) handles the storage and retrieval of objects from the database automatically. By using an object model InterMine queries benefit from inheritance, for example the *Gene* and *Exon* classes are both subclasses of *SequenceFeature*. When querying for SequenceFeatures (representing any genome feature) both Genes and Exons will be returned automatically.

We can see how see how inheritance is represented in the database:

- One table is created for each class in the data model.
- Where one class inherits from another, entries are written to both tables. For example:

```
biotestmine#  select * from gene limit 5;
```

The same rows appear in the *sequencefeature* table:

```
biotestmine#  select * from sequencefeature limit 5;
```

All classes in the object model inherit from *InterMineObject*. Querying the *intermineobject* table in the database is a useful way to find the total number of objects in a Mine:

```
biotestmine#  select count(*) from intermineobject;
```

All tables include an *id* column for unique ids and a *class* column with the actual class of that object. Querying the *class* column of *intermineobject* you can find the counts of different objects in a Mine:

```
biotestmine#  select class, count(*) from intermineobject group by class;
```

A technical detail: for speed when retrieving objects and to deal with inheritance correctly (e.g. to ensure a *Gene* object with all of its fields is returned even if the query was on the *SequenceFeature* class) a serialised copy of each object is stored in the *intermineobject* table. When queries are run by the ObjectStore they actually return the ids of objects - these objects are may already be in a cache, if not the are retrieved from the *intermineobject* table.

Loading Genome Data from GFF3 and FASTA

We will load genome annotation data for *P. falciparum* from PlasmoDB

- genes, mRNAs, exons and their chromosome locations - in GFF3 format
- chromosome sequences - in FASTA format

Data integration

Note that genes from the gff3 file will have the same *primaryIdentifier* as those already loaded from UniProt. These will merge in the database such that there is only one copy of each gene with information from both data sources. We will load the genome data then look at how data integration in InterMine works.

First, look at the information currently loaded for gene *PFL1385c* from UniProt:

```
biotestmine=#  select * from gene where primaryIdentifier = 'PFL1385c';
```

GFF3 files

GFF3 is a standard format use to represent genome features and their locations, each line represents one feature and has nine tab-delimited columns:

```
MAL1    ApiDB   gene    183057  184457  .       -       .       ID=gene.46311;description=hypothetica
MAL1    ApiDB   mRNA    183057  184457  .       +       .       ID=mRNA.46312;Parent=gene.46311
MAL1    ApiDB   exon    183057  184457  .       -       0       ID=exon.46313;Parent=mRNA.46312
```

col 1: “seqid” an identifier for a ‘landmark’ on which the current feature is locatated, in this case ‘MAL1’, a ‘’P. falciparum” chromosome.

col 2: “source” the database or algorithm that provided the feature

col 3: “type” a valid Sequence Ontology term defining the feature type - here *gene* or *mRNA*

col 4 & 5: “start” and “end” coordinates of the feature on the landmark in col 1

col 6: “score” an optional score, used if the feature has been generated by an algorithm

col 7: “strand” ‘+’ or ‘-‘ to indicate the strand the feature is on

col 8: “phase” for *CDS* features to show where the feature begins with reference to the reading frame

col 9: “attributes” custom attributes to describe the feature, these are name/value pairs separated by ‘;’. Some attributes have predefined meanings, relevant here:

- *ID* - identifier of feature, unique in scope of the GFF3 file
- *Name* - a display name for the feature
- *Parent* - the *ID* of another feature in the file that is a parent of this one. In our example the *gene* is a *Parent* of the *mRNA*.

A dot means there is no value provided for the column.

The files we are loading are from PlasmoDB and contain *gene*, *exon* and *mRNA* features, there is one file per chromosome. Look at an example:

```
$ less /data/malaria/genome/gff/MAL1.gff3
```

The GFF3 source

InterMine includes a parser to load valid GFF3 files. The creation of features, sequence features, locations and standard attributes is taken care of automatically.

Other *gff3* properties can be configured in the *project.xml* The properties set for *malaria-gff* are:

gff3.seqClsName = Chromosome the ids in the first column represent *Chromosome* objects, e.g. MAL1

gff3.taxonId = 36329 taxon id of malaria

gff3.dataSourceName = PlasmoDB the data source for features and their identifiers, this is used for the DataSet (evidence) and synonyms.

gff3.seqDataSourceName = PlasmoDB the source of the seqids (chromosomes) is sometimes different to the features described

gff3.dataSetTitle = PlasmoDB P. falciparum genome a DataSet object is created as evidence for the features, it is linked to a DataSource (PlasmoDB)

You can also configure GFF properties in the gff.config file. See *GFF3* for details.

To deal with any specific attributes or perform custom operations on each feature you can write a handler in Java which will get called when reading each line of GFF. For malaria gff we need a handler to switch which fields from the file are set as *primaryIdentifier* and *symbol/secondaryIdentifier* in the features created. This is to match the identifiers from UniProt, it is quite a common issue when integrating from multiple data sources.

From the example above, by default: *ID=gene.46311;description=hypothetical%20protein;Name=PFA0210c* would make *Gene.primaryIdentifier* be *gene.46311* and *Gene.symbol* be *PFA0210c*. We need *PFA0210c* to be the *primaryIdentifier*.

Look at the *malaria-gff.properties* file - there are two properties of interest:

```
# set the source type to be gff
have.file.gff=true

# specify a Java class to be called on each row of the gff file to cope with attributes
gff3.handlerClassName = org.intermine.bio.dataconversion.MalariaGFF3RecordHandler
```

The property file has specified a Java class to process the GFF file, MalariaGFF3RecordHandler. This code changes which fields the *ID* and *Name* attributes from the GFF file have been assigned to.

Loading GFF3 data

Now execute the *malaria-gff* source by running this command:

```
# load the GFF data
~/git/biotestmine $ ./gradlew integrate -Psource=malaria-gff --stacktrace
```

This will take a few minutes to run. Note that this time we don't run *buildDB* as we are loading this data into the same database as UniProt. As before you can run a query to see how many objects of each class are loaded:

```
$ psql biotestmine
biotestmine#  select class, count(*) from intermineobject group by class;
```

FASTA files

FASTA is a minimal format for representing sequence data. Files comprise a header with some identifier information preceded by '>' and a sequence. At present the InterMine FASTA parser loads just the first entry in header after > and assigns it to be an attribute of the feature created. Here we will load one FASTA file for each malaria chromosome. Look at an example of the files we will load:

```
$ less /data/malaria/genome/fasta/MAL1.fasta
```

The type of feature created is defined by a property in *project.xml*, the attribute set defaults to *primaryIdentifier* but can be changed with the *fasta.classAttribute* property. The following properties are defined in *project.xml* for *malaria-chromosome-fasta*:

fasta.className = org.intermine.model.bio.Chromosome the type of feature that each sequence is for

fasta.dataSourceName = PlasmoDB the source of identifiers to be created

fasta.dataSetTitle = PlasmoDB chromosome sequence a DataSet object is created as evidence

fasta.taxonId = 36329 the organism id for malaria

fasta.includes = MAL*.fasta files to process

This will create features of the class *Chromosome* with *primaryIdentifier* set and the *Chromosome.sequence* reference set to a *Sequence* object. Also created are a *DataSet* and *DataSource* as evidence.

Loading FASTA data

Now run the *malaria-chromosome-fasta* source by running this command:

```
# load FASTA data
~/git/biotestmine $ ./gradlew integrate -Psource=malaria-chromosome-fasta --stacktrace
```

This has integrated the chromosome objects with those already in the database. In the next step we will look at how this data integration works.

Data Integration

Data integration in BioTestMine

The sources *uniprot-malaria* and *malaria-gff* have both loaded information about the same genes. Before loading genome data we ran a query to look at the information UniProt provided about the gene "PFL1385c":

```
biotestmine=# select id, primaryidentifier, secondaryidentifier, symbol, length , chromosomeid, chro
    id    | primaryidentifier | secondaryidentifier | symbol | length | chromosomeid | chromosomeloca
----------+-------------------+---------------------+--------+--------+--------------+---------------
83000626 | PFL1385c          |                     | ABRA   |        |              |
(1 row)
```

Which showed that UniProt provided *primaryIdentifier* and *symbol* attributes and set the *organism* reference. The *id* was set automatically by the ObjectStore and will be different each time you build your Mine.

Running the same query after *malaria-gff* is added shows that more fields have been filled in for same gene and that it has kept the same id:

```
biotestmine=# select id, primaryidentifier, secondaryidentifier, symbol, length , chromosomeid, chro
    id    | primaryidentifier | secondaryidentifier | symbol | length | chromosomeid | chromosomeloca
----------+-------------------+---------------------+--------+--------+--------------+---------------
83000626 | PFL1385c          | gene.33449          | ABRA   |   2232 |     84017653 |             840
(1 row)
```

This means that when the second source was loaded the integration code was able to identify that an equivalent gene already existed and merged the values for each source, the equivalence was based on *primaryIdentifier* as this was the field that the two sources had in common.

Note that *malaria-gff* does not include a value for *symbol* but it did not write over the *symbol* provided by UniProt, actual values always take precedence over null values (unless configured otherwise).

Now look at the organism table:

```
biotestmine=# select * from organism;
genus | taxonid | species | abbreviation |    id    | shortname | name |                class
-------+---------+---------+--------------+----------+-----------+------+-----------------------------
      |   36329 |         |              | 83000003 |           |      | org.intermine.model.genomic
(1 row)
```

Three sources have been loaded so far that all included the organism with *taxonId* 36329, and more importantly they included objects that reference the organism. There is still only one row in the organism table so the data from three sources has merged, in this case *taxonId* was the field used to define equivalence.

How data integration works

Data integration works by defining keys for each class of object to describe fields that can be used to define equivalence for objects of that class. For the examples above:

- *primaryIdentifier* was used as a key for *Gene*

- *taxonId* was used as a key for *Organism*

For each *Gene* object loaded by *malaria-gff* a query was performed in the *biotestmine* database to find any existing *Gene* objects with the same *primaryIdentifier*. If any were found fields from both objects were merged and the resulting object stored.

Many performance optimisation steps are applied to this process. We don't actually run a query for each object loaded, requests are batched and queries can be avoided completely if the system can work out no integration will be needed.

We may also load data from some other source that provides information about genes but doesn't use the identifier scheme we have chosen for *primaryIdentifier* (in our example *PFL1385c*). Instead it only knows about the *symbol* (*ABRA*), in that case we would want that source to use the *symbol* to define equivalence for *Gene*.

Important points:

- A *key* defines a field or fields of a class that can be used to search for equivalent objects
- Multiple primary keys can be defined for a class, sources can use different keys for a class if they provide different identifiers
- One source can use multiple primary keys for a class if the objects of that class don't consistently have the same identifier type
- *null* - if a source has no value for a field that is defined as a primary key then the key is not used and the data is loaded without being integrated.

Integration Keys in BioTestMine

The keys used by each source are set in the source's *resources* directory.

- uniprot-malaria
- malaria-gff

The key on *Gene.primaryIdentifier* is defined in both sources, that means that the same final result would have been achieved regardless of the order in the two sources were loaded.

These *_keys.properties* files define keys in the format:

```
Class.name_of_key = field1, field2
```

The *name_of_key* can be any string but you must use different names if defining more than one key for the same class, for example in *uniprot_keys.properties* there are two different keys defined for *Gene*:

```
Gene.key_primaryidentifier = primaryIdentifier
Gene.key_secondaryidentifier = secondaryIdentifier
```

It is better to use common names for identical keys between sources as this will help avoid duplicating database indexes. Each key should list one or more fields that can be a combination of *attributes* of the class specified or *references* to other classes, in this cases there should usually be a key defined for the referenced class as well.

The *tracker* table

A special *tracker* table is created in the target database by the data integration system. This tracks which sources have loaded data for each field of each object. The data is used along with priorities configuration when merging objects but is also useful to view where objects have come from.

- Look at the columns in the tracker table, *objectid* references an object from some other table
- Query tracker information for the objects in the examples above:

```
select distinct sourcename from tracker, gene where objectid = id and primaryidentifier = 'PFL1385c';

select objectid, sourcename, fieldname, version from tracker, gene where objectid = id and primaryide

select distinct sourcename from tracker, organism where objectid = id;
```

Updating Organism and Publication Information

Organisms and publications in InterMine are loaded by their taxon id and PubMed id respectively. The *entrez-organism* and *update-publications* sources can be run at the end of the build to examine the ids loaded, fetch details via the NCBI Entrez web service and add those details to the Mine.

Fetching organism details

You will have noticed that in previous sources and in *project.xml* we have referred to organisms by their NCBI Taxonomy id. These are numerical ids assigned to each species. We use these for convenience in integrating data, the taxon id is a good unique identifier for organisms whereas names can come in many different formats: for example in fly data sources we see: *Drosophila melanogaster*, *D. melanogaster*, Dmel, DM, etc.

Looking at the *organism* table in the database you will see that the only column filled in is *taxonid*:

```
$ psql biotestmine
biotestmine#  select * from organism;
```

From the root *biotestmine* directory run the *entrez-organism* source:

```
# load organism data
~/git/biotestmine $ ./gradlew integrate -Psource=entrez-organism --stacktrace
```

This should only take a few seconds. This source does the following:

- runs a query in the production database for all of the taxon ids
- creates an NCBI Entrez web service request to fetch details of those organisms
- converts the data returned from Entrez into a temporary Items XML file
- loads the Items XML file into the production database

Now run the same query in the production database, you should see details for ‘’P. falciparum” added:

```
$ psql biotestmine
biotestmine#  select * from organism;
```

As this source depends on organism data previously loaded it should be one of the last sources run and should appear at the end of *<sources>* in *project.xml*.

Fetching publication details

Publications are even more likely to be cited in different formats and are prone to errors in their description. We will often load data referring to the same publication from multiple sources and need to ensure those publications are integrated correctly. Hence we load only the PubMed id and fetch the details from the NCBI Entrez web service as above.

Several InterMine sources load publications:

```
biotestmine#  select count(*) from publication;
biotestmine#  select * from publication limit 5;
```

Now run the *update-publications* source to fill in the details:

```
~/git/biotestmine $ ./gradlew integrate -Psource=update-publications --stacktrace
```

As there are often large numbers of publications they are retrieved in batches from the web service.

Now details will have been added to the *publication* table:

```
biotestmine#  select * from publication where title is not null limit 5;
```

As this source depends on publication data previously loaded it should be one of the last sources run and should appear at the end of <*sources*> in *project.xml*.

Post Processing

Post-processing steps are run after all data is loaded, they are specified as <*post-process*> elements in *project.xml*.

Some of these can only be run after data from multiple sources are loaded. For example, for the Malaria genome information we load features and their locations on chromosomes from *malaria-gff* but the sequences of chromosomes from *malaria-chromosome-fasta*. These are loaded independently and the *Chromosome* objects from each are integrated, neither of these on their own could set the sequence of each *Exon*. However, now they are both loaded the *transfer-sequences* post-process can calculate and set the sequences for all features located on a *Chromosome* for which the sequence is known.

Some post-process steps are used to homogenize data from different sources or fill in shortcuts in the data model to improve usability - e.g. *create-references*.

Finally, there are post-process operations that create summary information to be used by the web application: *summarise-objectstore*, *create-search-index* and *create-autocomplete-indexes*.

BioTestMine Post Processing

The following <*post-process*> targets are included in the BioTestMine *project.xml*.

Run queries listed here before and after running the post-processing to see examples of what each step does.

create-references This fills in some shortcut references in the data model to make querying easier. For example, *Gene* has a collection of *transcripts* and *Transcript* has a collection of *exons*. *create-references* will follow these collections and create a *gene* reference in *Exon* and the corresponding *exons* collection in *Gene*.

```
biotestmine#  select * from exon limit 5;
```

The empty *geneid* column will be filled in representing the reference to gene.

Execute the *create-references* postprocess by running this command:

```
# execute create-references postprocess
~/git/biotestmine $ ./gradlew postprocess -Pprocess=create-references
```

transfer-sequences The sequence for chromosomes is loaded by *malaria-chromosome-fasta* but no sequence is set for the features located on them. This step reads the locations of features, calculates and stores their sequence and sets the *sequenceid* column. The *sequenceid* column for this exon is empty:

```
biotestmine# select * from exon where primaryidentifier = 'exon.32017';
```

Execute the *transfer-sequences* postprocess by running this command:

```
# execute transfer-sequences postprocess
~/git/biotestmine $ ./gradlew postprocess -Pprocess=transfer-sequences
```

After running *transfer-sequences* the *sequenceid* column is filled in.

do-sources Each source can also provide code to execute post-process steps if required. This command loops through all of the sources and checks whether there are any post-processing steps configured. There aren't any for the sources we are using for BioTestMine but you should always include the *do-sources* element.

summarise-objectstore*, *create-search-index* & *create-autocomplete-index These generate summary data and search indexes used by the web application, see *Keyword Search* for details.

Execute the *summarise-objectstore* postprocess by running this command:

```
# execute transfer-sequences postprocess
~/git/biotestmine $ ./gradlew postprocess -Pprocess=summarise-objectstore
```

You must have Solr installed and running for the indexes to be populated correctly.

Install SOLR

Download Solr binary package and extract it to any place you like. Inside */solr-7.2.1* directory start the server with this command:

```
# Starts the server instance on port 8983
solr-7.2.1 $ ./bin/solr start
```

Initialising Search Indexes

To create a Intermine collection for search process, run this command inside the solr directory.

```
# Initialises the search index
solr-7.2.1 $ ./bin/solr create -c biotestmine-search
```

To create a Intermine collection for autocomplete process, run this command inside the solr directory.

```
# Initaliases the autocomplete index
solr-7.2.1 $ ./bin/solr create -c biotestmine-autocomplete
```

These are empty search indexes. These will be populated by the *create-search-index* & *create-autocomplete-index* postprocesses.

See *Solr* for details.

Execute the *create-search-index* and *create-autocomplete-index* postprocesses by running this command:

```
# execute create-search-index and create-autocomplete-index postprocesse
~/git/biotestmine $ ./gradlew postprocess -Pprocess=create-search-index
~/git/biotestmine $ ./gradlew postprocess -Pprocess=create-autocomplete-inde
```

Building a Mine with a Perl script

So far we have created databases, integrated data and run post-processing with individual gradle tasks. Alternatively InterMine has a Perl program called *project_build* that reads the *project.xml* definition and runs all of the steps in sequence. The script has the option of creating snapshots during the build at specified checkpoints.

Build complete BioTestMine

To build BioTestMine using the *project_build* script, first download the script:

```
# download the script
~/git/biotestmine $ wget https://raw.githubusercontent.com/intermine/intermine-scripts/master/project
# make executable
~/git/biotestmine $ chmod +x project_build
```

Run the *project_build* script from your *biotestmine* directory:

```
~/git/biotestmine $ ./project_build -b -v localhost ~/biotestmine-dump
```

This will take ~15-30mins to complete.

Note: If you encounter an "OutOfMemoryError", you should set your $GRADLE_OPTS variable, see *Troubleshooting tips*

Deploying the web application

You can deploy a web application against your newly built database.

Configure

In the *~/.intermine* directory, update the webapp properties in your biotestmine.properties file. Update the following properties:

- tomcat username and password
- superuser username and password

UserProfile

The userprofile database stores all user-related information such as username and password, tags, queries, lists and templates.

1. Configure

Update your biotestmine.properties file with correct information for the *db.userprofile-production* database:

```
db.userprofile-production.datasource.serverName=DB_SERVER
db.userprofile-production.datasource.databaseName=userprofile-biotestmine
db.userprofile-production.datasource.user=USER_NAME
db.userprofile-production.datasource.password=USER_PASSWORD
```

2. Create the empty database:

```
$ createdb userprofile-biotestmine
```

3. Build the database:

```
# creates the empty tables
~/git/biotestmine $ ./gradlew buildUserDB
```

You only need to build the userprofile database once.

Warning: The buildDB and buildUserDB commands rebuild the database and thus will delete any data.

Deploying the webapp

Before deploying the biotestmine webapp, you need to configure tomcat. See *Tomcat* for configuration details.

Run the following command to release your webapp:

```
# deploy the webapp (tomcat must be running)
~/git/biotestmine $ ./gradlew cargoDeployRemote
```

If you make changes, redeploy your webapp with this command:

```
# REdeploy the webapp (tomcat must be running)
~/git/biotestmine $ ./gradlew cargoReDeployRemote
```

Using the webapp

Navigate to http://localhost:8080/biotestmine to view your webapp. The path to your webapp is the *webapp.path* value set in biotestmine.properties.

Next

Now that you have a database and a working webapp, you'll want to know how to add your own logo, pick a colour scheme, modify how data is displayed etc. Our *webapp tutorial* is a detailed guide on how to customise all parts of the InterMine web application.

Help

Gradle

Anytime you run *./gradlew* and something bad happens, add the *–stacktrace* or *–debug* options.

This will give you more detailed output and hopefully a more helpful error message.

Logs

If the error occurs while you are integrating data, the error message will be in the *intermine.log* file in the directory you are in.

If the error occurs while you are browsing your webapp, the error message will be located in the Tomcat logs: *$TOMCAT/logs*.

Contact us!

Please contact us if you run into problems. We have a discord server, twitter and a developer mailing list.

1.2.2 Tutorial - Configure your InterMine webapp!

This tutorial aims to cover the basics of configuring an InterMine webapp.

Overview

In general, customisation of InterMine is accomplished by updating the appropriate configuration file and redeploying the webapp. A few features are updated via tagging as well. See *Guide to Customising your Web Application* for the full documentation on the webapp.

Note: You should have completed the previous tutorial and have successfully deployed the webapp.

This tutorial is intended to give a general idea of what you can customise in InterMine and how to do it. We're going to go through each section of the webapp and give step by step instructions on how to configure different parts of the page. This is a detailed tutorial and should take you a few hours to complete – however it is not meant to be comprehensive. Where topics aren't covered, there are links provided for more information. If you have a question that you don't see answered, try searching the documentation or taking a look at the index. Intermine has an active developer's *Mailing list* as well.

Tomcat

You will need to have Tomcat running for this tutorial.
If your webapp is under heavy usage or development, Tomcat may run out of memory. See *Tomcat* for details on how to update your settings to adjust the amount of memory available to Tomcat.

General Layout

Each web page in InterMine has the same header and footer. The header contains everything at the top of the page, including the navigation tabs and the keyword search. The footer contains the contact form and InterMine logo.

Let's start configuring our mine by updating these common sections of our web application.

Header

Logo First, let's update the logo of your site. The logo should be 45x43 and named *logo.png*, for example:

1. Copy your image into this directory: *./webapp/src/main/webapp/model/images/logo.png*.
2. Deploy your webapp with this command:

```
$ ./gradlew cargoRedeployRemote
```

3. Refresh your browser

You should see your new logo in the top left corner of your webapp. If you don't, try clearing your browser's cache.

clean If your changes are still not being reflected in your webapp, add the *clean* target:

```
$ ./gradlew clean; ./gradlew cargoRedeployRemote
```

This removes all temporary directories so you are certain your new files are being used.

See `/system-requirements/software/gradle/index` for a list of all available Gradle tasks.

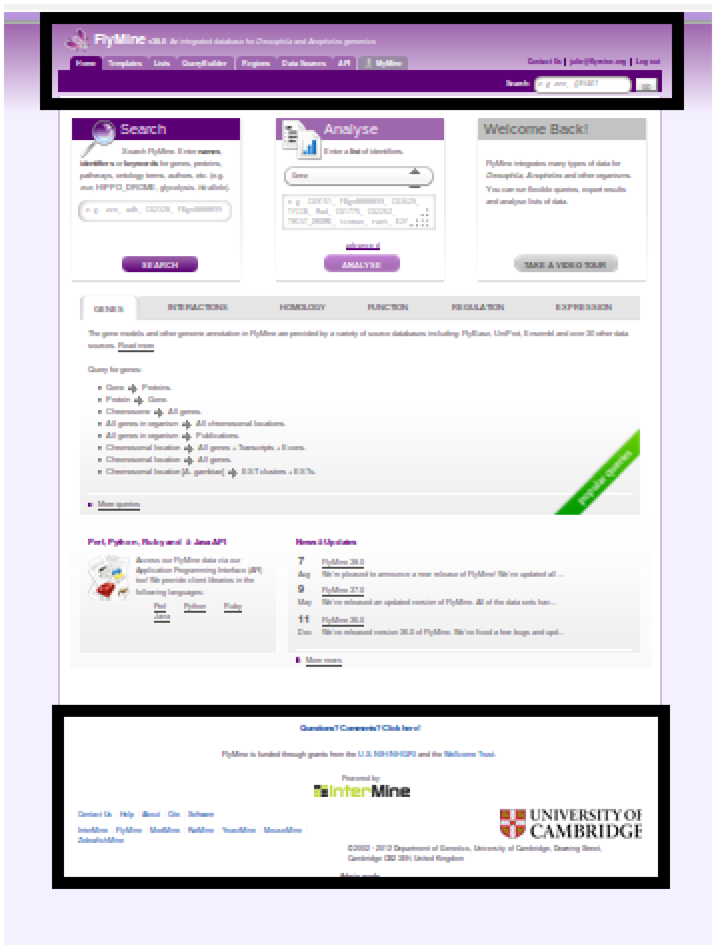


Figure 1.1: Header and footer of FlyMine website

Figure 1.2: FlyMine's logo

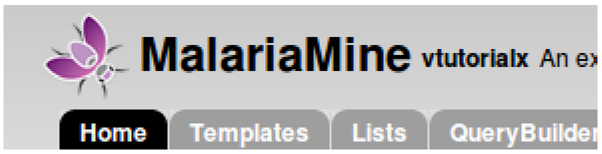


Figure 1.3: Updated logo

Subtitle and Release version Next to the name of your mine in the header is the release version and subtitle for your mine:

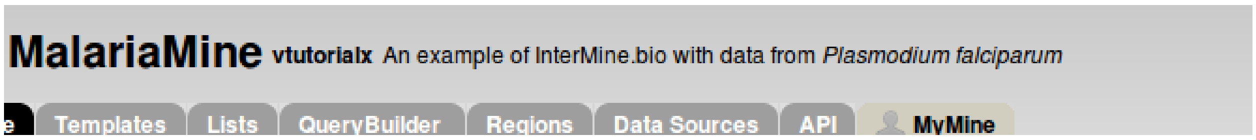

MalariaMine vtutorialx An example of InterMine.bio with data from *Plasmodium falciparum*
Templates
Lists
QueryBuilder
Regions
Data Sources
API
MyMine

Figure 1.4: Title, release version and subtitle

These values are set in *Database and Web application* file. This is the same properties file you updated in the previous tutorial. The subtitle and release versions are populated by the properties *project.subTitle* and *project.releaseVersion*, respectively. Update these properties to a different value and redeploy your webapp using the commands given above. Once you have successfully released your webapp, you should see your new subtitle.

1. Open the properties file in your favourite text editor.

```
$ emacs ~/.intermine/biotestmine.properties
```

2. Update the values of the subtitle and release version. Save your work.

```
# text that appears in the header and elsewhere
project.title=BioTestMine
project.subTitle=An example of InterMine.bio with data from <i>Plasmodium falciparum</i>
project.releaseVersion=tutorialx
```

3. Redeploy your webapp

```
$ ./gradlew cargoRedeployRemote
```

4. Navigate to your mine's home page and see the updated values: http://localhost:8080/biotestmine

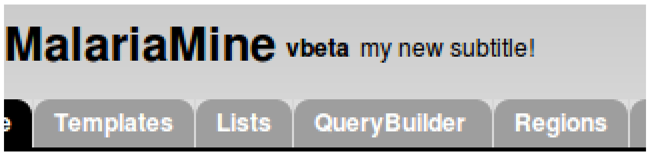


Figure 1.5: Updated release version and subtitle

That's it! Well done! The majority of mine configuration will be accomplished this way - update a property in a text file and redeploy the webapp.

See *Database and Web application* for the full list of properties this file controls.

How do I know which property to change? Now you know how to change properties and configure your mine. How then do you know *which* property to change? There are a few resources available to you:

Guide to Customising your Web Application A detailed listing of everything you can configure in the InterMine webapp. It's grouped by InterMine webpage, e.g. Home Page, so you should be able to find what you need easily.

Google The search for this site is quite good although you can still use Google, e.g. here's a Google search for help with logos.

Table of Contents / Index On the upper right hand corner of every page are links to the Index and the table of contents. Both are fairly comprehensive.

Ask us! A quick email to the dev *Mailing list* usually proves to be quite helpful too.

Show all properties

You can also see and edit the values of every property set for your mine.

1. Log in as the superuser for your mine. (See *Website Admin* for details on how to do this.)
2. Change the last part of the URL in your browser to be *showProperties.do*, e.g. http://localhost:8080/biotestmine/showProperties.do

This lists of all properties that are used in your webapp. You can update the values for each property and instantly see how the webapp is changed, without worrying about breaking anything. (The changes only last for that session, to permanently change a value you'll need to update the appropriate config file.)

Keyword Search

InterMine's keyword search uses a Lucene-based index created at build-time. Every field in the database is indexed unless you configure a table or column to be skipped. You can also configure facets / categories to help your users mine the search results. See *Keyword Search* for details on how to configure the keyword search.

The first search

When the first search is executed after a webapp is released, the search index is:

1. Retrieved from the database
2. Written to temp files
3. Loaded into memory for use by the webapp

This can take up to a minute. Our release scripts include a command to run this search so that the index is preloaded.

The search box contains example identifiers to help your users know which types of search terms to use. To update the default value, set the *quicksearch.example.identifiers* property in the *web.properties* file. Redeploy your webapp to see your changes.

Note: The Lucene index can become quite large, depending on the size of the database. FlyMine's index is ~2G, so make certain you have plenty of room.

Footer

The footer is positioned at the bottom of every page in the InterMine webapp. It contains the contact link and the funding message.

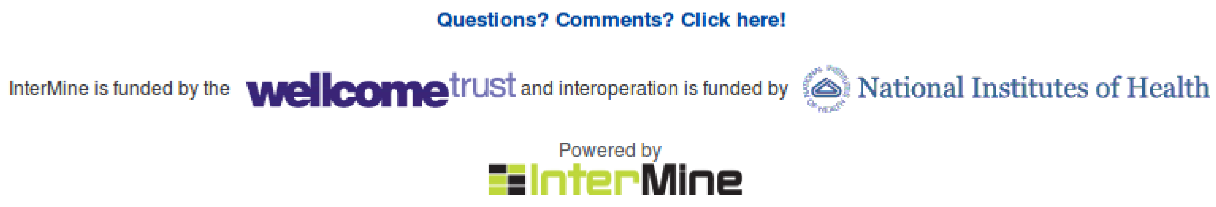

Questions? Comments? Click here!
InterMine is funded by the wellcome trust and interoperation is funded by National Institutes of Health
Powered by InterMine

Figure 1.6: Funding message in footer

To update the funding message, change the *funding* property in *Text and messages*. Redeploy your webapp to see your changes.

```
# Model specific internationalisation properties
# this file merges with InterMineWebApp.properties

funding = InterMine is funded by the <a href="http://www.wellcome.ac.uk/" target="_new" title="Wellco
```

Here is the bit of code in footer.jsp that renders that message: https://github.com/intermine/intermine/blob/dev/intermine/webapp/main/re

The *model.properties* is the third configuration file you've edited today, there are four main files that control most of the behaviour in your InterMine webapp.

InterMine properties files

~/.intermine/biotestmine.properties	database and webapp names and locations. includes passwords and shouldn't be in source control.
web.properties	webapp behaviour, e.g. link outs, tabs on home page
model.properties	text displayed on webapp, e.g. error messages
webconfig-model.xml	webapp functionality, e.g. custom export types, widgets, data display

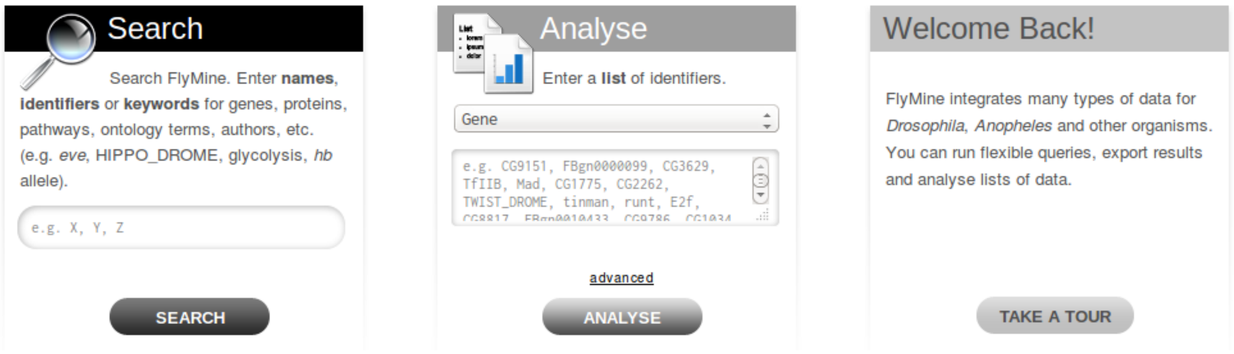
See *General Layout* for more details on how to update the header, footer and colour scheme of your InterMine webapp. Next we'll customise your home page.

Home page

Most everything on the home page is customisable. You can edit the text and set which RSS news feed to use. If you want something very different, you can create and use your own home page.

Boxes

You can customise the text in the three boxes that appear on the top of the home page. Let's edit the example given in the middle box marked *Analyse*.



Search FlyMine. Enter **names**, **identifiers** or **keywords** for genes, proteins, pathways, ontology terms, authors, etc. (e.g. *eve*, HIPPO_DROME, glycolysis, *hb* allele).

Enter a **list** of identifiers.

e.g. CG9151, FBgn0000099, CG3629, TfIIB, Mad, CG1775, CG2262, TWIST_DROME, tinman, runt, E2f,

FlyMine integrates many types of data for *Drosophila*, *Anopheles* and other organisms. You can run flexible queries, export results and analyse lists of data.

Figure 1.7: Three boxes at the top of the home page

Notice the text box already has an example, *e.g. X, Y, Z*. This is the default example and it's set by *begin.listBox.example* in an InterMine properties file, *global.web.properties*.

Add *begin.listBox.example* to your mine's *Features* file and redeploy your webapp to see your changes.

See *Home page* for more details on how to update your home page.

1.2.3 Quick Start

This guide will show you how to create a *BioTestMine* InterMine instance. You will need all the dependencies listed in *Software*

Note: Please see *Tutorial*, a detailed guide to integrating data with InterMine.

1. Get the software

Clone the repository

```
~/git/ $ git clone https://github.com/intermine/biotestmine.git
```

You now have an InterMine! The *tutorial* goes into detail about the various files that comprise an InterMine.

2. Add a mine properties file

Your InterMine uses a properties file for database usernames and passwords, let's create that file now.

Make an intermine directory in your home directory.

```
# change to be in your home directory
~/git $ cd
# make an intermine directory
~ $ mkdir .intermine
```

Copy the properties file from the git repository to your local InterMine directory.

```
~/.intermine $ cp ~/git/biotestmine/data/biotestmine.properties
```

Now update your new properties files with the values correct for your InterMine. Replace PSQL_USER and PSQL_PWD with your postgres username and password.

See *Database and Web application* for details on this file and what each property means.

3. Set up your search index (optional)

Solr handles the keyword search in InterMine. See *Solr* for details on how to set Solr up for your mine.

If you skip this step, your mine will work fine but the keyword search will fail.

4. Build + deploy your webapp

Now run the build!

```
~/git/biotestmine $ ./setup.sh
```

Your build (depending on your server) will take a few minutes to run.

Next steps

Congratulations! You now have an InterMine! Visit it at localhost:8080/biotestmine.

Next you will want to:

- learn how to use the InterMine webapp
- *customise your mine*
- *add your own data sources*
- *join the intermine mailing list*

1.2.4 Create Your Own InterMine!

This guide will show you how to create a new InterMine. You will need all the dependencies listed in *Software*.

Note: These instructions assume you have taken the *Tutorial*, a detailed guide to integrating data with InterMine.

See *Quick Start* to run our test InterMine - BioTestMine.

1. Run a script to create your InterMine

Download the script.

```
~/git/ $ git clone https://github.com/intermine/intermine-scripts.git
```

Execute the script to generate your InterMine instance. Here we are using *TigerMine* but of course you would use your mine name here.

```
~/git/ $ ~/git/intermine-scripts/make_mine TigerMine
```

You will see a message like: *created /home/$USER/git/tigermine directory for tigermine*.

You now have an InterMine! The *tutorial* goes into detail about the various files that comprise an InterMine.

2. Add a mine properties file

Your InterMine uses a properties file for database usernames and passwords, let's create that file now.

Make an intermine directory in your home directory.

```
# change to be in your home directory
~/git $ cd
# make an intermine directory
~ $ mkdir .intermine
```

Copy the properties file you created in the tutorial.

```
~/.intermine $ wget https://github.com/intermine/biotestmine/blob/master/data/biotestmine.properties
```

Rename the file to match your Mine.

```
~/.intermine $ mv biotestmine.properties tigermine.properties
```

Now update your new properties files with the values correct for your InterMine. You'll want to update the details for your InterMine databases, you'll create those in the next step.

See *Database and Web application* for details on this file and what each property means.

3. Create databases

Just as in the demo, you will create your InterMine databases.

```
# move into your mine directory
~ $ cd ~/git/tigermine
# create the database for your mine data
~/git/tigermine $ createdb tigermine
~/git/tigermine $ createdb items-tigermine
# create the database for user information
~/git/tigermine $ createdb userprofile-tigermine
```

Note: These database names should match the ones you added to your mine.properties file in the previous step.

These databases are empty. We'll populate the main database in the following steps, but let's put some default information in our user database now.

```
# create the empty tables for the user database, plus add the superuser
~/git/tigermine $ ./gradlew buildUserDB
```

4. Update project file

The data loaded into your mine is controlled by the *project.xml* file located in the root of your mine directory. See *Project XML* for an in depth description of this file.

InterMine has a few dozen libraries for popular data sources you can use. See *Data Source Library* for the full list. Select one of the data sources and add it to your project XML file. Don't forget to download the data too.

For example, *NCBI - Entrez gene* loads gene information from the NCBI. Download the data files listed, then add the given project XML entry to your own mine's project XML file, like so:

```
<source name="ncbi-gene" type="ncbi-gene">
    <property name="src.data.dir" location="/$DATA/ncbi" />
    <property name="organisms" value="9606" />
</source>
```

See *Writing your own data source* if you want to load your own data into your mine.

You can also add "postprocesses" to your build, these are tasks that run after the database build, tasks to build the search index for example. Here are common ones you might want to include:

```
<post-processing>
  <post-process name="do-sources" />
  <post-process name="create-attribute-indexes" />
  <post-process name="summarise-objectstore" />
  <post-process name="create-autocomplete-index" />
  <post-process name="create-search-index" />
</post-processing>
```

See *Post processing* for details on what postprocesses do.

5. Set up your search index (optional)

Solr handles the keyword search in InterMine. See *Solr* for details on how to set Solr up for your mine.

If you skip this step, your mine will work fine but the keyword search will fail.

6. Build + deploy your webapp

Now run the build!

```
# download the script
~/git/tigermine $ wget https://raw.githubusercontent.com/intermine/intermine-scripts/master/project_b
# make executable
~/git/tigermine $ chmod +x project_build
```

Run the *project_build* script from your *mine* directory:

```
~/git/tigermine $ ./project_build -b localhost /data/tigermine-build
```

See *project_build script* for more on the *project_build* script.

Your build (depending on your sources) will take a few minutes to run. Once that is done, deploy your webapp. Make sure tomcat is running.

```
# deploy your webapp to tomcat
~/git/tigermine $ ./gradlew cargoDeployRemote
# if you have already deployed once, you will want to run this command instead:
~/git/tigermine $ ./gradlew cargoRedeployRemote
```

See `/system-requirements/software/gradle/index` for more on Gradle.

Next steps

Congratulations! You now have an InterMine! Visit it at localhost:8080/tigermine. (replace *tigermine* with the name your chose for your mine)

Next you will want to:

- learn how to use the InterMine webapp
- *customise your mine*
- *add your own data sources*
- *join the intermine mailing list*

1.2.5 Testmine

This is an InterMine used for testing new features, and for continuous integration tests on Travis. Its tables include: Employee, Company, Department. The mine does not contain biological data.

To start a testmine, run the setup script:

```
testmine $ ./setup.sh
```

It uses your UNIX username if you haven't set the PSQL_USER, PSQL_PWD ENV variables. The script copies the testmodel.properties file into your home *.intermine* directory.

There are different targets to load data:

- insertData - Loads basic data, e.g. EmployeeA, EmployeeB
- loadsadata - Loads basic data set and testmodel_extra_data.xml
- enormocorp - Loads basic data set, testmodel_extra_data.xml, and testmodel_enormo_data.xml

The setup script runs *loadsadata*.

```
# run to see which tasks are available for you
testmine $ ./gradlew tasks
```

1.2.6 InterMine Tests

Continuous Integration

We run all our tests on every commit using the Continous Integration service Travis. You can do the same for your fork:

- Log in to Travis-CI with your GitHub account.
- Enable your fork of intermine for Travis builds.

All the tests will be run on every change you make, and you will be notified of errors by email.

Setting up a Local Test Environment

After getting the source code for InterMine and ensuring you have all of the required prerequisites, the next step is to try the tests to confirm that everything runs well in your environment.

We also recommend looking at the files that run our continous integration tests for examples of how this can be automated:

- *config/travis/init.sh*
- *config/travis/run.sh*

Running the core tests

Create databases

Create blank databases required by the tests named: *unittest*, *truncunittest*, *fulldatatest*, *flatmodetest*, *notxmltest*. See PostgresBasics and introduction to some Postgres commands.

```
$ for db in unittest truncunittest fulldatatest flatmodetest notxmltest; do createdb $db; done
```

Update properties file

You need to set up a properties file to provide database details to the test code. In your home directory create a file called *intermine-test.properties* and update the server name, database names, and database username and password. You can use different database names as long as the actual database name used to create the database and the *db.xxx.datasource.databaseName* value match.

```
# super user
superuser.account=test

# common properties

os.query.max-time=10000000
os.query.max-limit=100000
os.query.max-offset=10000000
os.queue-len=100
```

```
# testing properties

db.notxmlunittest.datasource.serverName=localhost
db.notxmlunittest.datasource.databaseName=notxmltest
db.notxmlunittest.datasource.user=USERNAME
db.notxmlunittest.datasource.password=SECRET_PASSWORD

db.truncunittest.datasource.serverName=localhost
db.truncunittest.datasource.databaseName=truncunittest
db.truncunittest.datasource.user=USERNAME
db.truncunittest.datasource.password=SECRET_PASSWORD

db.flatmodeunittest.datasource.serverName=localhost
db.flatmodeunittest.datasource.databaseName=flatmodetest
db.flatmodeunittest.datasource.user=USERNAME
db.flatmodeunittest.datasource.password=SECRET_PASSWORD

db.fulldatatest.datasource.serverName=localhost
db.fulldatatest.datasource.databaseName=fulldatatest
db.fulldatatest.datasource.user=USERNAME
db.fulldatatest.datasource.password=SECRET_PASSWORD

db.userprofile-test.datasource.serverName=localhost
db.userprofile-test.datasource.databaseName=userprofile-test
db.userprofile-test.datasource.user=USERNAME
db.userprofile-test.datasource.password=SECRET_PASSWORD

db.unittest.datasource.serverName=localhost
db.unittest.datasource.databaseName=unittest
db.unittest.datasource.user=USERNAME
db.unittest.datasource.password=SECRET_PASSWORD
```

Run the tests

```
# in intermine
$ ./gradlew test
```

View results

The HTML test report will be created in the build directory, eg. *intermine/objectstore/test/build/test/results/index.html*

Pull requests are not accepted without passing tests, and we have Travis set up to run tests on every commit. We keep the tests at a 100% pass rate at all times.

Running the bio tests

InterMine includes a *bio* project which contains specific code for biological data and parsers for many data formats. To run tests on this code you need to set up another properties file and create some more databases.

Create databases

Create blank databases called *bio-test* and *bio-fulldata-test* (as above you can use different names as long as they match the *db.xxx.datasource.databaseName* values. For example:

```
$ createdb bio-test
$ createdb bio-fulldata-test
```

Update properties file

Set up a properties file to provide database details to the test code. In *.intermine* create a file called *intermine-bio-test.properties* and configure the server name, database names, and database username and password.

```
os.default=os.production-client

# common properties

os.query.max-time=10000000
os.query.max-limit=100000
os.query.max-offset=10000000
os.queue-len=100

# testing properties

db.bio-fulldata-test.datasource.serverName=localhost
db.bio-fulldata-test.datasource.databaseName=bio-fulldata-test
db.bio-fulldata-test.datasource.user=USERNAME
db.bio-fulldata-test.datasource.password=SECRET_PASSWORD

db.bio-test.datasource.serverName=localhost
db.bio-test.datasource.databaseName=bio-test
db.bio-test.datasource.user=USERNAME
db.bio-test.datasource.password=SECRET_PASSWORD
```

Build the databases

Build database tables automatically generated from the bio model by running the following in *bio*:

```
$ ./gradlew builddb
```

Run the tests

Execute the tests, in *bio* run:

```
$ ./gradlew test
```

Run a single test

You can also run a test for an individual source by using this syntax:

```
# in bio
$ ./gradlew bio-model:test
```

The test results will be located at *bio/model/test/build/test/results/index.html*. You can also run these as JUnit tests directly from Eclipse or Intellij.

1.3 InterMine

1.3.1 InterMine JARs

InterMine JARs are published on JCenter: https://bintray.com/intermineorg

To put these on your classpath, add the correct dependencies, e.g.

Maven

```
<dependency>
    <groupId>org.intermine</groupId>
    <artifactId>intermine-api</artifactId>
    <version>4.0.1</version>
    <type>pom</type>
</dependency>
```

Gradle

```
compile 'org.intermine:intermine-api:4.0.1'
```

1.3.2 Upgrading InterMine

See our blog for details on each of the InterMine releases. You can view the release notes and associated tickets on GitHub too.

Upgrade Instructions

For non-disruptive releases, you can upgrade your mine by incrementing your version number in your mine's *gradle.properties* file:

```
# example -- flymine's gradle.properties
systemProp.imVersion=4.0.0
systemProp.bioVersion=4.0.0
```

To get patch updates automatically, use the plus (+) notation:

```
# example -- flymine's gradle.properties
systemProp.imVersion=4.0.+
systemProp.bioVersion=4.0.+
```

Read more: *InterMine Versioning Policy* and *InterMine Development Roadmap*

InterMine 4.1.3

This is a non-disruptive release.

It contains a small batch of bug fixes.

InterMine 4.1.2

This is a non-disruptive release.

InterMine 4.1.1

This is a non-disruptive release.

It contains some bug fixes related to ncbi-gff bio source and few improvements from ThaleMine.

You can build your mine using Gradle wrapper 4.9. To update the version, run the following command in your InterMine instance directory:

```
cd flymine
./gradlew wrapper --gradle-version 4.9
```

See our blog post for more details (https://intermineorg.wordpress.com/2019/10/29/intermine-4-1-1-patch-release/).

InterMine 4.1.0

This is a non-disruptive release.

Galaxy integration has been improved; you should remove the galaxy related properties from the web.properties file to benefit of it.

Integration with ELIXIR AAI has been included.

Gradle wrapper updated to the 4.9 version.

Some bug fixes.

See our blog post for more details (https://intermineorg.wordpress.com/2019/09/24/intermine-4-1-0/)

InterMine 4.0.1

Restore Strains to core data model.

InterMine 4.0.0

DataSet.licence was added to the data model. To update to this new data model for this release, you'll want to rebuild your database and redeploy your webapp.

To enable the structured data added to the web pages in format of JSON-LD, you should set the property *markup.webpages.enable* to true in the web.properties file.

To configure the new URLs used in the "share" button, you should specify the keys in the class_keys.properties file.

See our blog post for more details on how to configure and use the new features to make your mine to be more FAIR.

InterMine 3.1.2

This is a non-disruptive release.

InterMine 3.1.1

This is a non-disruptive release.

InterMine 3.1.0

The class *Strain* was added to the core InterMine data model in this release.

- You will need to rebuild your database with the new model to release a new webapp.
- If you do have Strains in your data, you might think about using the core data classes now available.

```
<!-- core.xml -->
<class name="Strain" extends="BioEntity" is-interface="true">
    <attribute name="annotationVersion" type="java.lang.String"/>
    <attribute name="assemblyVersion" type="java.lang.String"/>
    <collection name="features" referenced-type="SequenceFeature" reverse-reference="strain" />
</class>

<class name="SequenceFeature" extends="BioEntity" is-interface="true">
    <!-- snip -->
    <reference name="strain" referenced-type="Strain"  reverse-reference="features" />
</class>

<class name="Organism" is-interface="true">
    <!-- snip -->
    <collection name="strains" referenced-type="Strain"/>
</class>
```

To update to use the new InterMine release:

- Change your mine's *gradle.properties* file to *3.1.+*.

  ```
  # example -- flymine's gradle.properties
  systemProp.imVersion=3.1.+
  systemProp.bioVersion=3.1.+
  ```

- Change your data sources' *gradle.properties* file to *3.1.+*.

  ```
  # example -- flymine-bio-sources gradle.properties
  systemProp.imVersion=3.1.+
  systemProp.bioVersion=3.1.+
  ```

InterMine 3.0.0

This release adds Solr to InterMine. To upgrade, you will need to rebuild your database and install Solr.

To Upgrade

1. Change your mine's *gradle.properties* file to *3.0.+*. If you have data sources, change the version they use too.

   ```
   # example -- flymine's gradle.properties
   systemProp.imVersion=3.0.+
   systemProp.bioVersion=3.0.+
   ```

2. Install Solr

   Solr

3. Configure Solr

   Keyword Search

4. Rebuild your database.

Specifically the postprocesses that build the search index.

You should then be able to deploy your webapp as normal, with the new and improved search.

InterMine 2.+

InterMine 2.0 is a disruptive release and is not backwards compatible. This means that databases, webapps and code from previous releases will need to be updated to work with the new InterMine release. Below are detailed instructions on how to migrate your InterMine to the new build system.

> **Warning:** If you have custom InterMine code, your changes will likely not work as expected after the upgrade. Please contact us and we can help you migrate your edits to the new system.

Please contact us if you have any questions or concerns! We have a mailing list or you can contact us directly via email or our discord channel (chat.intermine.org). If you are having difficulties, we can also arrange a skype call to walk through any problems together. Please make sure your code is public, e.g. GitHub, so we can help test!

Gradle

InterMine now uses Gradle to manage dependencies and to build and run InterMine. Please see `Gradle Quick Start` for useful Gradle commands and `Gradle FAQs` for help with common questions and errors.

See the Gradle blog post for details as to why we made this change.

Maven

You will need Maven installed. We use Maven to manage mine-specific InterMine dependencies, including your mine-specific data parsers.

```
# for Ubuntu
sudo apt-get install maven
```

You do not need to install Gradle locally. Instead, use the Gradle wrapper provided.

Remove InterMine code

Previously you had to download and compile InterMine. Now, instead, you'll be using the compiled InterMine JARs available via Maven. This means you should remove all InterMine code from your mine repositories. Your mine repositories should only contain your mine (webapp and dbmodel) and your mine's custom data sources.

If you have your mine and bio/sources in your InterMine checkout, instead of in their own repository, you'll have to separate them out.

What you want to end up with:

- FlyMine - https://github.com/intermine/flymine/ (MUST be the name of your mine)
- FlyMine specific data sources - https://github.com/intermine/flymine-bio-sources

Options to separate out your mine repo:

1. You can copy over your directories directly. Don't do this! You'll lose your history.

```
# don't do this
~/git $ cp intermine/flymine flymine; cd flymine
~/git/flymine $ git init; git add *; git commit -am "initial commit"
```

2. Instead, use *git filter-branch* command. Follow the directions on how to move a directory to a new repository and keep your history in GitHub.

You should not have any core InterMine code locally.

New directory structure

InterMine has switched to use the standard Maven directory structure.

```
src/main/java
src/main/resources
src/test/java
src/test/resources
```

You will have to run two migration scripts to move your current mine over to this new layout – one script for your mine and one for your mine's data parsers. The migration scripts are located in the intermine-scripts repository.

```
~/git $ git clone https://github.com/intermine/intermine-scripts.git
```

Migrate Mine webapp to New directory structure Run "migrateMine" script to move your mine over to the new directory system. You might want to create a new *gradle* branch for testing.

```
~/git/intermine-scripts/gradle-migration/mine $ ./migrateMine.sh ~/git/flymine
```

Migrate Data Sources to New directory structure Run the "migrateBioSources" script to move your sources over to the new directory system.

```
~/git/intermine-scripts/gradle-migration/data-sources $ ./migrateBioSources.sh ~/git/flymine-bio-sour
```

Run this command to put your sources on the classpath and therefore available to the database build:

```
# not part of the upgrade process. You will install every time you make a change
~/git/flymine-bio-sources $ ./gradlew install --stacktrace
```

This task builds the JARs and places them on your classpath in *~/.m2/repository*.

Note the command is *./gradlew* instead of *gradle*. Use the provided Gradle wrapper instead of locally installed Gradle.

You will have to *install* your sources every time you update the source code to update the JAR being used by the build.

Previously the data model was merged from all data sources' additions XML file. This is no longer true. Since each source is in its own JAR now, the data model is self-contained. Therefore if you reference a class in your data parser, it must be present in the additions file. Alternatively, you can specify a single data model file that will be merged into each source:

```
// [in build.gradle in root of your mine bio/sources directory, e.g. flymine-bio-sources]
// uncomment to specify an extra additions file for your bio-sources
// this file will be merged with the additions file for each data source
// and included in each source JAR.
//bioSourceDBModelConfig {
//    # file should live in your mine's bio/sources directory
//    globalAdditionsFile = "MY-MINE_additions.xml"
//}
```

Update config

1. Remove *<property name="source.location" location="../bio/sources/"/>* from your project XML file
2. Set *GRADLE_OPTS* instead of *ANT_OPTS*
   - Use the same parameters.
   - Append *-Dorg.gradle.daemon=false* to prevent daemons from being used.
3. Update project XML for some sources
   - *SO* source's location has been updated to be: *<property name="src.data.file" location="so.obo" />*
   - *Protein2ipr* source has a new attribute: *<property name="osAlias" value="os.production"/>*
   - *intermine-items-xml-file* isn't a valid value for "type" anymore. Use the project name instead.
   - *src.data.dir* can only have a *location* attribute. *src.data.dir* cannot have a *value* attribute.
   - Change the location of the generated files for *entrez-organism* and *update-publications* data sources to be *organisms.xml* and *publications.xml* (instead of in the *build* directory)
4. InterPro data file needs to be updated. The file incorrectly references *interpro.dtd* when you should have the full path instead.
   - Update interpro.xml
   - *<!DOCTYPE interprodb SYSTEM "ftp://ftp.ebi.ac.uk/pub/databases/interpro/interpro.dtd">*
   - I asked InterPro to fix but they said no. Maybe you could ask too?
   - See https://github.com/intermine/intermine/issues/1914 for the discussion.
5. Update each data source's additions file to be correct. Alternatively you can use the *extraAdditionsFile* (see previous section).
6. *PostprocessUtil.java* moved to the *bio* package, so you maybe have to update your import to be *import org.intermine.bio.util.PostProcessUtil;*.

Please see `Gradle Quick Start` for details on Gradle and common Gradle commands and `Gradle FAQs` for help with common questions and errors.

Data Model

- Syntenic Regions have been added to the data model
- Protein.molecularWeight is now a Float instead of an Integer
- GO evidence codes now have a name and URL
- OntologyAnnotation can now annotate any InterMine object, as long as that class inherits *Annotatable*
- Sequence Ontolgy has been updated to the latest version
- Organism.taxonId is a String instead of an Integer.

See the Model Changes blog post for details.

You have may to update your data sources and queries to match the new data model.

Dependencies

Software dependency requirements have been updated to the latest versions. This is so we can get rid of legacy code and make use of new features.

```
Java SDK 8
Tomcat 8.5.x
Postgres 9.3+
```

You will get errors if you use older versions. e.g. If you use Java 7, you will get this error: *Caused by: java.security.NoSuchProviderException: no such provider: SunEC*

API changes

We are making some non-backwards compatible changes to our API. These three end points have a parameter called *xml* which holds the XML query. We are going to rename this parameter to be *query* (as we now accept JSON queries!) to match the syntax of all the other end points.

```
/query/upload
/template/upload
/user/queries (POST)
```

Please update any code that references these end points.

Pre-InterMine 2.0 Upgrade Instructions

To pull changes in your local repository and merge them into your working files:

```
$ git pull upstream
```

If you host a copy of the *CDN*, then you should also pull in changes from that repository.

Upgrade to InterMine 1.6

The core model of InterMine has changed in release 1.1 so you may encounter more errors than usual.

update integration keys You may need to update your integration keys if they are using a class or field that's been changed.

update custom converter If you are storing data using a class or field that's been changed, you will have to change your code to use the new model. See below for the complete list of model changes.

template queries You will have to update your templates to use the new model

interaction viewer The cytoscape tool uses the new model - will not work until you build a database with the new code

Interactions

class	old	new
Interaction	gene1	participant1
	gene2	participant2
	relationshipType (Term)	relationshipType (String)
InteractionDetail	allInteractors (Gene)	allInteractors (Interactor)
Interactor	–	stoichiometry
	InteractionDetail.role1	role
	InteractionDetail.type	type

Protein Domains

class	old	new
ProteinDomain	proteins	proteinDomainRegions
Protein	proteinDomains	proteinDomainRegions
ProteinDomainRegion	–	start
	–	end
	–	identifier
	–	database

Upgrade to InterMine 1.4

There are no model changes, but we've added some new features that require an update.

We've added a new fancy connection pool, you should see a performance improvement. However you do need to update some configuration files.

Postgres config file

The number of database connections required will depend on your usage. 100 connections is the default and should be okay for production webapps. However each webapp reserves 20 connections so on your dev machines it may be wise to raise the maximum quite a bit.

postgresql.conf

max_connections=250

$MINE properties files

in your $MINE directory:

default.intermine.integrate.properties

set
db.production.datasource.maxConnections=20
db.common-tgt-items.datasource.maxConnections=5
and for each database replace
db.production.datasource.class=org.postgresql.ds.PGPoolingDataSource
(or any other db pooling class)
with these 2 lines
db.production.datasource.class=com.zaxxer.hikari.HikariDataSource db.production.datasource.dataSourceClassName=org.postgresq

default.intermine.webapp.properties

set
db.production.datasource.maxConnections=20
and for each database replace
db.production.datasource.class=org.postgresql.ds.PGPoolingDataSource
(or any other db pooling class)
with these 2 lines
db.production.datasource.class=com.zaxxer.hikari.HikariDataSource db.production.datasource.dataSourceClassName=org.postgresq

Any other data source you use should be set to five connections, raised to ten if you encounter problems, e.g. the build failing with an error like so:

Error message

Caused by: org.postgresql.util.PSQLException: FATAL: connection limit exceeded for non-superusers

Or this (See #912)

Error message

Unable to get sub-ObjectStore for Translating ObjectStore

See *HikariCP and InterMine settings* for details.

InterMine-model Refactor

The metadata package has moved from to InterMine-model. If you have custom data sources that use InterMine Utils, you may have to update your code to reflect the new location. Your IDE should be able to do this for you.

Tomcat

Add *clearReferencesStopTimerThreads* to your $TOMCAT/conf/context.xml file, so it should look like so:

```
<Context sessionCookiePath="/" useHttpOnly="false" clearReferencesStopTimerThreads="true">
...
</Context>
```

Upgrade to InterMine 1.3.x

This code will work with any webapp and database created with InterMine 1.3+.

Upgrade to InterMine 1.3

- Remove all duplicate entries from web.xml
- Model changes:
  - DataSet now has a publication reference
  - AnnotationExtension has been moved from GOAnnotation to GOEvidence.

Also, we have changed our GO parser a bit. Each line in a gene annotation file now corresponds with an Evidence object. In prior releases, each Evidence object was unique, e.g. only a single evidence code per gene / GO term pair.

Upgrade to InterMine 1.2.1

If you have your own home page (begin.jsp), you must manually make this change: 501e221

This is a fix for the keyword search - when users submit a blank search form, see Issue #329

There are no model or configuration changes in this release.

Upgrade to InterMine 1.2

The core data model has not been changed, so you should be able to release a webapp using InterMine 1.2 code without making any changes.

Upgrade to InterMine 1.1

The core model of InterMine has changed in release 1.1 so you may encounter more errors than usual.

update integration keys You may need to update your integration keys if they are using a class or field that's been changed.

update custom converter If you are storing data using a class or field that's been changed, you will have to change your code to use the new model. See below for the complete list of model changes.

template queries You will have to update your templates to use the new model

interaction viewer Widget uses the new model - will not work until you build a database with the new code

Model Changes

Updated to latest version of Sequence Ontology, 2.5

old	new
Comment.text	Comment.description
Gene.ncbiGeneNumber	–
–	Gene.description
–	Gene.briefDescription

Interactions

class	old	new
Interaction	gene	gene1
	interactingGenes	gene2
	type	details.type
	role	details.role1
	–	details.role2
	name	details.name
	shortName	–
InteractionRegion	primaryIdentifier	–
	name	–

Gene Ontology

class	old	new
GOAnnotation	withText	evidence.withText
	with	evidence.with
	–	annotationExtension
OntologyTerm	–	crossReferences [3]

Identifiers

We have several [wiki:Homologue new homologue data converters] available in this InterMine release. However, some of these new data sources use Ensembl IDs. If you want to load the model organism database identifier instead (important for interoperation with other InterMines), you should use the Entrez Gene ID resolver:

1. Download the identifier file - ftp://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz
2. Unzip the file
3. Add the path to properties file:

```
# in ~/.intermine/MINE_NAME.properties
resolver.entrez.file=/DATA_DIR/ncbi/gene_info
```

Configuration Updates

Web services uses the *webapp.baseurl* property to run queries, so be sure this is the valid URL for your mine. Otherwise you will get an "Unable to construct query" error on the query results page.

```
# in ~/.intermine/MINE_NAME.properties
# used by web services for running queries, needs to be valid
webapp.baseurl=http://localhost:8080
```

1.3.3 InterMine Development Roadmap

InterMine is a noncommercial, free software project, and as such there is no formal list of feature requirements required for development.

We ensure that all new features committed to InterMine are thoroughly vetted by our community of contributors and committers.

Upcoming minor releases

The InterMine project aims to make at least one minor release every quarter. If it becomes necessary due to an important bugfix or security issue, more releases will be made, so this list should be considered a minimum.

The current schedule for upcoming releases is:

- Fall 2019 (4.1.0)
- Winter 2019 (4.2.0)
- Spring 2020
- Summer 2020

[3] used for Uberon

Next major release

The next major release of InterMine is planned to be InterMine 5.0.0.

While there are no formal requirements for each InterMine release, there are several places you can look to find out more information on upcoming features:

- InterMine mailing list
- InterMine community calls
- Blog
- InterMine releases
- InterMine roadmap

1.3.4 InterMine Versioning Policy

Version Numbering

InterMine uses semantic versioning:

MAJOR	incompatible API changes
MINOR	functionality added in a backwards-compatible manner
PATCH	backwards-compatible bug fixes

InterMine releases a new major version containing new features about once a year. Each major version receives bug fixes and, if need be, security fixes that are released at least once every three months in what we call a "minor release." For more information on the minor release schedule, you can view the minor release `</intermine/roadmap>`.

If the release team determines that a critical bug or security fix is too important to wait until the regularly scheduled minor release, it may make a release available outside of the minor release schedule.

We always recommend that all users run the latest available minor release.

Upgrading

Major versions often change the data model or the InterMine API. These changes are often complex, so we do not maintain backward compatibility. A database rebuild is required. We also recommend reading the *upgrading* section of the major version you are planning to upgrade to.

Upgrading to a minor release does not normally require a database rebuild; you can stop your webapp, update your InterMine version number, and redeploy your webapp. For some releases, manual changes may be required to complete the upgrade, so always read the release notes before upgrading.

While upgrading will always contain some level of risk, InterMine minor releases fix only frequently-encountered bugs, security issues, and blocking problems to reduce the risk associated with upgrading. For minor releases, the **community considers not upgrading to be riskier than upgrading**.

1.3.5 Contribution Guide

This document sets out the development processes for those contributing to the InterMine code base. It specifically refers to the main application code-base, but these practices should be employed in an ideal world on all code bases.

There is no distinction between the processes that developers should follow internally or externally - all code contributions, whether from core team members or outside contributers, should be treated the same.

Branches

There are branches in the InterMine GitHub repository with special meaning:

`master` The current public release. External users should clone this branch and receive a stable, supported and well-documented application that works to all specifications.

`dev` The working branch. Features are merged onto this branch for integration testing. Not guaranteed to be stable.

Setting Up a Development Environment

Development does not happen on the master or dev branch. The recommended practice is to fork the intermine repo and maintain development branches in your own repository.

Developing a Feature

Code contributions should be discrete units of code. They should do one thing (be that fix a bug or add a feature) and not be code dumps. Ideally they should refer to existing issues in the *InterMine issue tracker*. Let's say we want to develop a new feature - discussed in issue `#12345: We should be better wombles and recycle everything` - then we would do the following:

1. Checkout the current head of *dev* from upstream.
2. Branch *dev*, naming the branch something descriptive like `womblier`.
3. Checkout the new branch.
4. Commit, commit, commit. Using detailed commit messages.
5. Push changes to your fork.
6. When you are satisfied that we have reached a sufficiently wombly state of being, then create a new pull request requesting that the head of `you/womblier` be merged into `intermine/dev`.

At any point in the above process you can merge switch to work on another branch and then come back. It is probably a good idea to regularly merge the head of `intermine/dev` into `you/womblier`, especially if development is taking a long time. These merges should probably be `rebase` merges.

Hot fix branches (serious bugs that are critical fixes to the current release) should be branched from `master` rather than `dev`, and their pull requests should likewise be for `master`.

The Role of The Release Manager

The release manager's role is to ensure this all happens. They are the only person permitted to push into `master` and `dev`. All code contributions for these branches must pass review by the release manager before they can be merged.

The process for reviewing an merging a pull request is as follows:

1. Read the commits and review the code for style and standards. Request any changes from the developer before proceeding. The criteria for acceptance is:

- Passing unit test for new code (if applicable)
- Passes all tests – according to Travis
- Documentation (if applicable)
- Single purpose
- Detailed commit messages

- Well commented code
- Checkstyle

2. Fetch and checkout the new feature branch
3. Merge the target branch (`master` or `dev`) into the feature branch. If there are any conflicts push the pull-request back to the developer for resolution.
4. Perform necessary automated and manual testing to verify that this branch is valid.
5. Checkout the current head of `intermine/dev` and merge the feature branch into it.
6. Push `dev` to the intermine repo.

Release Process

Once all pull requests and tickets for a specific milestone are tested and complete, the release manager merges the *dev* branch onto the *master* branch tagging the merge with the milestone's label. The release notes are available on the Releases page, and announcments are posted on twitter and the mailing lists and discussed in detail on the community calls.

1.3.6 How to set up your InterMine webapp to use https

You will need to use a CDN delivering https content (see *Performance*), for example https://cdn.intermine.org

Set the corresponding entry in 'global.web.properties', for example

```
head.cdn.location = https://cdn.intermine.org
```

You can also override this property by setting it directly in your `mine.properties` file.

Note: If you are moving your existing mine to https, please take care of updating also the following properties in the same `mine.properties` file:

- project.sitePrefix
- webapp.deploy.url
- webapp.baseurl

If you are using your own jbrowse server, this will now need to be served through https as well, and you will need to adjust also the property:

- jbrowse.install.url

Tomcat requirements

You should add a configuration to your tomcat server.xml in the Engine section, specifying the address of your proxy:

```
<Valve className="org.apache.catalina.valves.RemoteIpValve"
     protocolHeaderHttpsValue="https"
     remoteIpHeader="x-forwarded-for"
     requestAttributesEnabled="true"
     internalProxies="\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"
     protocolHeader="x-forwarded-proto" />
```

It is also good practice to limit access to tomcat port only to the host's loopback address (localhost):

```
<Connector port="8080" protocol="HTTP/1.1"
        address="127.0.0.1"
        connectionTimeout="20000"
        URIEncoding="UTF-8"
        redirectPort="8443" />
```

1.3.7 How to set up your InterMine environment on the Amazon Cloud

Where you should learn how to start your own MalariaMine web application on the Amazon Cloud. You could also use your InterMine Amazon instance to try building MalariaMine yourself or to build your own mine there.

Pre-requisites

You need an Amazon account: if you don't have one

- go to http://aws.amazon.com
- click on `Sign Up`
- follow the instructions

You will need to set up your key pair security mechanism (see for example step 7 below). Alternatively you will need your aws-access-key and your aws-secret-key to start your instance (not shown here).

Starting a new Instance

InterMine is publicly available on Amazon Cloud as an Image (AMI), with an AMI ID **ami-b1c7a9d8**.

The image contains a ready deployed MalariaMine.

1. sign in at http://aws.amazon.com
2. go to the EC2 management console AWS console https://console.aws.amazon.com/console/home –> EC2 console
3. if you don't have one, set up a security group which allows access at least to port
   - 22 (SSH)
   - 80 (HTTP)
   - 8080 (TOMCAT)

   you could set up also a few spare ones (20, 21, 8009).

Note: You can do this also during step 7, but **you cannot change the security group of an instance after starting it for the first time** (unless you use a VPC instance, see User Guide).

4. go to the IMAGES/AMI console
5. set the location on the top header (beside your username) to *US East (N. Virginia)*
6. set the filter to *Public Images* and search for **InterMine**
7. select *BasicIntermine* AMI (AMI ID = ami-b1c7a9d8)
8. launch (and configure) instance

- you can use all default options for the instance characteristics and details, but use the security group you created in step 3.
- when prompted, create a new key pair (`.pem` file), or use one that you already own.

9. go to the Instance console
10. select your new instance
11. when public DNS appears (after checks, a couple of minutes), you can open a terminal with

```
$ ssh -i your_pem_file ubuntu@the_instance_public_DNS
```

Starting an existing Instance

If you are using an existing Instance, you need to

1. sign in at http://aws.amazon.com
2. go to the EC2 console (see step 2 above)
3. go to the Instance console
4. select your instance
5. start your instance (Actions –> Start)

Working with Your Instance

Open a terminal in Your Instance

```
$ ssh -i your_pem_file ubuntu@the_instance_public_DNS
```

you will land in */home/ubuntu*

here you can find these relevant directories:

`git/intermine` the InterMine code base

`.intermine` with the properties file

`malaria` sources for building MalariaMine

Starting/stopping the existing MalariaMine web application

In `/webapp` you'll find tomcat6. You can start the webapp using this command:

```
$ ./start.sh
```

Your BioTestMine web application will be then available on

> http://the_instance_public_DNS:8080/malariamine

To stop the web application:

```
$ ./stop.sh
```

Redeploying MalariaMine

In /home/ubuntu/git/intermine/malariamine/webapp

```
$ ant -v default remove-webapp release-webapp
```

(Re)building MalariaMine

see http://intermine.readthedocs.org/en/latest/get-started/tutorial/

In */home/ubuntu/git/intermine/malariamine*

```
$ ../bio/scripts/project_build -b -v localhost ~/malariamine-dump
```

You can also follow all the steps in the build as illustrated in *Tutorial*

1.4 Data Model

1.4.1 Data Model Overview

InterMine uses an object-oriented data model, classes in the model and relationships between them are defined in an XML file. Depending on which data types you include you will need different classes and fields in the model, so the model is generated from a core model XML file and any number of additions files. These additions files can define extra classes to be added to the model and define extra fields for additional classes.

- Elements of the model are represented by Java classes and references between them.
- These Java classes map automatically to tables in the database schema.
- The object model is defined as an XML file, that defines classes, their attributes and references between classes.
- The Java classes and database schema are automatically generated from an XML file.

You can easily adapt InterMine to include your own data by creating new additions files, see the tutorial for a detailed walk though on how to do this.

Data source and Data set

Most data types in the InterMine core model have a reference to a “data set” and a corresponding “data source”.

Data source The origin of the data. Usually an organisation, e.g. UniProt, InterPro

Data set A set of results or data from a data source. e.g. InterPro GO Annotation data set

These data are meant to enable your users to easily trace the provenance of your data.

Organism

Include the *Organisms* data source in your build. Many of the tools available in InterMine assume this source will be loaded and expect a populated organism table.

Chromosome location

InterMine uses the -1 / 1 convention for strands.

Identifiers

All sequence features must have a non-NULL, unique identifier set for their *primaryIdentifier* field.

Sequence Ontology term

All sequence features should have a reference to the appropriate sequence ontology term. The Java data parsers do this for you automatically.

so_terms

Adding sequence ontology terms to the *so_terms* text file will add these classes to your data model.

- There is a mechanism for automatically generating a set of class definitions that reflect the structure of the SO.
  - Is-a relationships in the SO become subclass relationships in the model.
  - Part-of/member relationships in the SO become many-to-one or many-to-many relationships in the model (determined by the configs at the bottom of *so_terms*).
- Only the terms listed in *so_terms* become classes in the model.
  - In particular, a descendant class D and an ancestor class A may be included while none of the intervening classes (B and C) are.
  - The class generator takes care to make sure that D becomes a direct subclass of A and that it has whatever references/collections it would have inherited had B and C been included.
  - A particular example is transcript, which is four levels below sequence_feature in the SO, but Transcript is a direct subclass of SequenceFeature in the model. In addition, Transcript has a reference to Gene, inherited from the intervening SO term gene_member_region, which is omitted from the model.
- The model generated from *so_term* is augmented by the contents of intermine/bio/model/core.xml and intermine/bio/model/genomic_additions.xml (e.g., core.xml is where SequenceFeature is made a subclass of BioEntity).
- The generated model can be further augmented in the usual way by a source's source_additions.xml file and the global additions file.

Model Merging

The InterMine build system generates the data model by merging the following data files:

- core.xml
- genomic_additions.xml
- so_terms (see above)
- SOURCE_additions files for each data source listed in your project XML file
- *globalAdditionsFile* if specified

See *Model Merging* for details.

1.4.2 Model Description

A database stored using the InterMine system is object-oriented and it loads data defined by a model description. This model description is defined in a file, <MINENAME>_model.xml. This page describes the format of this file and its implications.

What the Model governs

The Model is a description of the class hierarchy that is expected to be stored in the database, so it includes a description of the classes and fields of the data that will be stored. The model will typically be used to generate Java code for those classes automatically. The auto-generated classes will be pure Java beans, with fields as described in the Model, with getters and setters. Each class can have any number of attributes (which store primitive data, like numbers, dates, and strings), references to other objects in the database, and collections of other objects in the database.

Since all objects in the database (except SimpleObjects) are instances of InterMineObject, which has a field called “id” which is unique, all objects in the database can be fetched individually by searching for that unique “id” value.

Naming conventions

The model expects standard Java names for classes and attributes. That is:

classes start with an upper case letter and be CamelCase. The name can’t include underscores or spaces.

fields (attributes, references, collections) should start with a lower case letter and be lowerCamelCase. The name shouldn’t include underscores or spaces.

It’s possible to specify friendly names that will be displayed in place of the actual java-ised name in the web interface.

The Model File Format

The Model is defined in an XML file, with only a few different tags. The document root tag is “<model>”, and contains a list of “<class>” tags, each of which describes a single class in the model. Class tags are not nested - the hierarchy is defined elsewhere, which allows multiple inheritance if necessary. All classes inherit all the fields of all its parent classes, so they should not be defined again.

The “<model>” Tag

The “<model>” tag has two attributes, which are mandatory:

name this is the name of the model. It should match the name of the file (that is, a model called “testmodel” must be in a file called “testmodel_model.xml”). A model can be fetched by name in Java by calling Model.getInstanceByName(String name) as long as this file is in the classpath.

package this is a unique path that defines the model.

The “<class>” Tag

name this is the name of the class. All the classes must be in the same Java package.

is-interface this must be “true” or “false”. If this is true, then the class is generated as a Java interface, which will allow multiple inheritance from this class. Objects can be created which are instances of an interface, by using dynamic code generation using Java reflection, and there is surprisingly little performance cost. If this is false, then the class will be a normal Java class, and instances will be normal Java objects. However, a Java class can only have one non-interface parent class. The main FlyMine Model is entirely interface. *In practice this field should always be set to true*

extends this is an optional space-separated list of other classes, specifying the parent classes of this class. Only one of these parents may be a non-interface. If this attribute is not present, then the parent of the class will be "InterMineObject", which is therefore indirectly the parent of all classes in the model (except SimpleObjects).

Inside the "<class>" tags are tags describing the fields of the class. These are "<attribute>", "<reference>", and "<collection>", none of which enclose any other XML tags. You should not define two fields with the same name for a particular class, taking into account that classes inherit all the fields of their parent classes. The InterMineObject class (which everything except SimpleObjects inherit) has a field called "id".

The "<attribute>" Tag

This tag defines a field in the class for storing primitive data, like numbers, dates, and Strings. It has two attributes:

name this is the name of the field, as it will appear in the Java class, and in queries.

type this is the type of data that can be stored in the field, and must be one of the following:

- boolean or java.lang.Boolean - this stores a simple "true" or "false" value. The first type is a primitive value with only those two possible values, whereas the latter type is the Java Boolean Object, which can contain a third value of "null".
- short or java.lang.Short - this stores a 16-bit signed integer value. Again, the latter type may also have a null value, as is the case with the rest of the numbers.
- int or java.lang.Integer - this stores a 32-bit signed integer value.
- long or java.lang.Long - this stores a 64-bit signed integer value.
- float or java.lang.Float - this stores a 32-bit floating-point number.
- double or java.lang.Double - this stores a 64-bit floating-point number.
- java.math.BigDecimal - this stores an arbitrary-precision floating point number. There is no Java primitive equivalent, so this field type may contain a null value.
- java.util.Date - this stores a date and time, with a resolution of one millisecond, or null.
- java.lang.String - this stores a portion of text of arbitrary length, or null.

The "<reference>" and "<collection>" Tags

The "<reference>" tag defines a field in the class for storing a reference to another object in the database. The "<collection>" tag defines a field in the class for storing a collection of references to other objects in the database. Both of these relationships may be unidirectional or bidirectional. If they are bidirectional, that means that there is an equivalent relationship in the referenced class that points in the reverse direction, and two relationships will agree on their contents. All referenced objects must be in the database for the references and collections to be valid. Both of these tags have several attributes:

name this is the name of the field, as it will appear in the Java class, and in queries.

referenced-type this is the class name of the class of object that is referenced by the reference, or present in the collection.

reverse-reference this is an optional name of a reference or collection in the referenced-type that is the reverse of this relationship. Specifying this turns the relationship into a bidirectional relationship.

Types of relationship

One to one relationship this is where a reference has a reverse-relationship that is also a reference. Use of these is discouraged, because they suffer from performance and consistency problems, and can possibly be better modelled by combining the two classes into one.

One to many relationship a collection has a reverse-relationship that is a reference. In this case you should always fill in the reference and leave the collection empty (it will be ignored).

e.g. Gene has a collection Transcripts and Transcript references one Gene, fill in Transcript.gene only.

Many to many relationship this is where a collection has a reverse-relationship that is a collection, or where a collection does not have a reverse-relationship. This type of collection can be altered from either side, and the changes will be observed in both sides.

In practice if one side is very large and the other smaller it is faster to populate the smaller collection.

e.g. Gene has a collection of Pathways and Pathway has a collection of Genes, fill in either Gene.pathways or Pathway.genes but not both. If Pathway.genes contains e.g. 20,000 items and Gene.pathways typically 100 items then it is faster to populate Gene.pathways.

Ontologies

It's possible to decorate your InterMine data model with ontology terms.

This isn't used anywhere (yet) but will be used in the future when we start generating RDF.

See for *How to add ontology terms to your model* for details.

A short example

```xml
<?xml version="1.0"?>
<model name="testing" package="org.intermine.model.bio">

  <class name="Protein" is-interface="true" term="http://semanticscience.org/resource/SIO_010043">
    <attribute name="name" type="java.lang.String" term="http://edamontology.org/data_2099"/>
    <attribute name="extraData" type="java.lang.String"/>
    <collection name="features"  referenced-type="NewFeature" reverse-reference="protein"/>
  </class>

  <class name="NewFeature" is-interface="true">
    <attribute name="identifier" type="java.lang.String"/>
    <attribute name="confidence" type="java.lang.Double"/>
    <reference name="protein" referenced-type="Protein" reverse-reference="features"/>
  </class>
</model>
```

For a more complete example, see FlyMine which covers all the features available in the model.

The Model defines the set of data that is searchable in the database. Other data can be written to the database, but only the classes and attributes that are defined in the model are searchable. So you may, if you wish, compile a Java class which inherits InterMineObject (to allow it to stored in the database) or some other class in the model, with extra fields, and store instances of that class in the database, but you will not be able to search for instances of that class, or for instances with a particular value for the field that is not in the model.

1.4.3 Using Class and Field Labels

The InterMine webapp, and to a limited extent web services, supports the use of labels for classes and fields. Unlabelled classes and fields are formatted for enhanced legibility.

The current system for determining a label is as follows:

1. If the class or field has a pre-set label, that is used
2. Otherwise the class or field name is

1. Split from its camel case parts as specified in Apache Commons StringUtils
2. Each part is given an initial upper-case
3. The parts are then joined by spaces

Handling paths is similar, except that the dots (".") between class and field names are replaced by right angle-brackets (">").

Examples

Before	After
ChromosomeLocation	Chromosome Location
shortName	Short Name
Organism	Organism
name	Name
Organism.shortName	Organism > Short Name

Well named fields and classes thus do not need explicit labelling.

Labels can be configured however in two ways, in order of precedence: #. Classes and fields can be configured individually. This configuration respects inheritance, and subclasses automatically inherit the field labels of their parents. #. Translation tables can be set up for classes and fields. These are for cases where *ALL* classes/fields with a certain name should be relabelled. Examples are *url* -> *URL*, which would otherwise be rendered as ''Url''. This is especially useful for acroynms.

Configuring classes and fields individually

To apply individual configuration, the file *webconfig-model.xml* needs to be edited, and a *label* attribute added to items you want to configure. eg:

```
<class className="org.intermine.model.bio.Allele" label="SOME CLASS LABEL">
    <fields>
      <fieldconfig fieldExpr="primaryIdentifier" label="SOME FIELD LABEL"/>
      <fieldconfig fieldExpr="symbol"/>
      <fieldconfig fieldExpr="alleleClass"/>
      <fieldconfig fieldExpr="organism.name" label="Organism"/>
    </fields>
</class>
```

This is most helpful in the case of compound field-expressions ("organism.name"), which can this be configured to display as just a single expression.

Configuring classes and fields globally

To configuring classes and fields globally, the mine needs to be made aware of properties files that hold the appropriate translations. Biological mines automatically get three of these files:

bio/webapp/src/main/webapp/WEB-INF/soClassName.properties used to generate readable names using the SO term a class represents

bio/webapp/src/main/webapp/WEB-INF/bioClassNames.properties used to map non-SO classes to readable names

bio/webapp/src/main/webapp/WEB-INF/bioFieldNames.properties uses to map field names to readable names

Additional files can be specified. Add the the following properties to your web.properties:

```
# put in your mines' web.properties file
web.config.classname.mappings.{SOME_ID}={RESOURCE_NAME}
web.config.fieldname.mappings.{SOME_ID}={RESOURCE_NAME}
```

All resources should be names relative to the WEB-INF directory where they will end up.

You can have as many additional files as you wish, but:

- They should all have a different id. If they do not, all but one will be silently ignored.
- They should not have configuration for the same class/field. If they do, and exception will be thrown on initialisation, and your webapp will not start.

Using these labels in your webapp

A new tag library is available to help with labelling. Add the following to the top of any jsp you write that you want to use labels in:

```
<%@ taglib uri="/WEB-INF/functions.tld" prefix="imf" %>
```

This library provides five functions, which expose static methods from the org.intermine.web.logic.WebUtil class:

formatPath(Path p, WebConfig wcf)

This function produces a fully configured string from an arbitrarily long path. eg: *<c:out value="${imf:formatColumnName(path, WEBCONFIG)}"/>*

formatPathStr(String s, InterMineAPI api, Webconfig wcf)

This function produces a fully configured string from an arbitrarily long path, where that path is represented as a string. *<c:out value="${imf:formatColumnName(pathString, INTERMINE_API, WEBCONFIG)}"/>*

formatField(Path p, Webconfig wcf)

This function produces a fully configured field name from the last field of an arbitrarily long path. eg: *<c:out value="${imf:formatField(path, WEBCONFIG)}"/>*

formatFieldStr(String s, InterMineAPI api, Webconfig wcf)

This function produces a fully configured field name from the last field of an arbitrarily long path, where that path is repr *<c:out value="${imf:formatFieldStr(pathString, INTERMINE_API, WEBCONFIG)}"/>*

formatFieldChain(String s, InterMineAPI api, Webconfig wcf)

This function produces a string of fully configured field names from all the fields in an arbitrarily long path, where that pa *<c:out value="${imf:formatFieldStr(pathString, INTERMINE_API, WEBCONFIG)}"/>*

The values *INTERMINE_API* and *WEBCONFIG* are automatically available within jsps at all times.

While it is possible to call the formatting methods of WebUtil directly from Java controllers, it is not advisable, from design principles, to do so. Labels are an aspect of presentation (the view) and thus not the responsibility of Java classes (the controllers). The only justifiable place to call presentation methods from is in action classes that directly return data to the user, eg. in webservices and ajax calls.

Using Labels in JavaScript

Pages in the InterMine webapp have a variable in the global scope named *$MODEL_TRANSLATION_TABLE*. This contains information on how all classes and their fields should be displayed.

To access its information, for classes:

```
var className = ??;
var displayName = $MODEL_TRANSLATION_TABLE[className].displayName;
```

And for fields of this class:

```
var fieldName = ??;
var fieldDisplayName = $MODEL_TRANSLATION_TABLE[className].fields[fieldName]
```

1.4.4 Querying over genomic ranges

InterMine includes functionality for querying features with overlapping genome coordinates. We have an index that is created on the *Location* table. This is used by a 'virtual' *SequenceFeature.overlappingFeatures* collection that is a *view* in the postgres database using the native Postgres index to find other features that overlap it.

In modMine (the InterMine for the modENCODE project) we also create *GeneFlankingRegion* features to represent specific distances upstream and downstream of genes to query for genes that are nearby other features.

Create the index

You need to create the index on the location table in your production database by adding the *create-location-range-index* post-process step to your *project.xml* file:

```
<post-process name="create-location-range-index"/>
```

Create the overlappingFeatures view

Create the *SequenceFeature.overlappingFeatures* view in the database. This allows you to query for any features that overlap any other types of features in the web interface or query API. Add the *create-overlap-view* post-process step, which needs to be located **after** *create-location-range-index* in your project XML file.

```
<post-process name="create-overlap-view" />
```

Now any queries on the *overlappingFeatures* collections will use this view and the new index.

1.4.5 Decorating your model with ontologies

It is possible to add ontolgy terms to the data types in your data model.

Why would you do this? Where is this used?

Adding an ontology term to a class will facilitate cross InterMine querying.

It can also enable cross-database analysis. Is the "gene" data type in MouseMine the same one as in the EBI?

We will use these ontologies in the future when we generate RDF.

How do you chose an ontology term?

We used an ontology search, then selected the most specific and accurate term available.

This is the search we used: https://bioportal.bioontology.org/search

We ended up selecting terms that were in the following ontologies:

- Sequence Ontology
- Semantic Science
- EDAM
- MeSH
- Dublin Core
- National Cancer Institute Thesaurus (US NIH)

How do you add an ontology term to the data model?

We've already added the terms to the core InterMine data model, and data types in the sequence ontology are updated automatically. You'll need to add ontology terms only to classes and attributes that you have added to your mine.

Once you have selected the correct ontology term, use the attribute *term* and add it to your data model. See the example below

An example additions.xml snippet with an ontology term

```
<?xml version="1.0"?>
<model name="testing" package="org.intermine.model.bio">
  <class name="Protein" is-interface="true" term="http://semanticscience.org/resource/SIO_010043">
    <attribute name="name" type="java.lang.String" term="http://edamontology.org/data_2099"/>
  </class>
</model>
```

For a more complete example, see FlyMine which covers many data types.

For a detailed description of the data model, see *Model Description*.

1.5 Database

1.5.1 Data Download Scripts

The DataDownloader system uses a plugin architecture to make it more straightforward to download data from arbitrary sources, and to add new sources to the system

Location

The system is a package located in our scripts repo here: https://github.com/intermine/intermine-scripts/tree/master/bio/DataDownloader

The package contains:

lib/DataDownloader Core libraries

lib/DataDownloader/Source Source Plugins

config configuration files

bin The executable launcher

Prerequisites

- Moose
- MooseX::ABC
- MooseX::FollowPBP
- MooseX::FileAttribute
- Net::FTP
- Log::Handler
- DateTime
- Module::Find
- Web::Scraper
- Ouch
- Number::Format
- PerlIO::gzip
- Perl6::Junction

If you are using Ubuntu (tested on 12.10), you can run the following command to install the packages:

```
$ sudo apt-get install libpath-class-perl libmoosex-types-path-class-perl liblog-handler-perl liblog-
```

Other perl modules need to be installed via CPAN:

```
$ cpan
cpan[1]> install MooseX::ABC
cpan[2]> install MooseX::FileAttribute
```

Data Source Configuration

To learn how to configure data sources of your mine, look here for examples:

DataDownloader/config

The yaml file of your mine is where data download script reads the instruction

Running

To run a set of data downloads, the following call should suffice:

```
perl DataDownloader/bin/download_data -e intermine
```

The Current working directory of the script is immaterial.

Specific sources can be run by naming them on the command line:

```
perl DataDownloader/bin/download_data -e intermine Uniprot GOAnnotation
```

Source names are case-sensitive. You can get a list of the available sources with the switch '–sources'.

Adding a new Source

A source is a class in the 'DataDownloader::Source' package that implements the following method:

- 'get_data': Get all the data for this source

And accepts the following arguments in its constructor:

- *data_dir => "dirname"* the name of a directory to put data in, preferably in a sub-directory.* *logger => Log::Handler* A logger to use to log error and debug messages.Exceptions may be thrown by a source at any time. They will be caught and logged. It is the source's responsibility to clean up after itself however.

A template for creating a source is available in the form of an abstract class all Sources are expected to inherit from. This class, *DataDownloader::Source::ABC* makes it simple to add straightforward source downloaders, and provides helpers to make it convenient to add complex ones.

A minimal source can be seen in the form of *bio/scripts/DataDownloader/lib/DataDownloader/Source/FlyAnatomyOntology.pm*:

```
package DataDownloader::Source::FlyAnatomyOntology;

use Moose;
extends 'DataDownloader::Source::ABC';

use constant {
    TITLE  => 'Fly Anatomy Ontology',
    DESCRIPTION => "Drosophila Anatomy ontology from FlyBase",
    SOURCE_LINK => "http://www.flybase.net/",
    SOURCE_DIR => 'ontologies/fly-anatomy',
    SOURCES => [{
        FILE   => 'fly_anatomy.obo',
        SERVER => 'http://obo.cvs.sourceforge.net/*checkout*/obo/obo/ontology/anatomy/gross_anatomy/a
    }],
};

1;
```

This source fully inherits the behaviour of the 'DataDownloader::Source::ABC' abstract class, and only adds configuration. In this case, it defines a set of constants that describe this source:

- 'TITLE': The human readable name of the source shown in log messages.
- 'DESCRIPTION': A Longer description of the data that appears in a version file.
- 'SOURCE_LINK': A link to the origin of the material that appears in the version file.
- 'SOURCE_DIR': The sub-directory under the 'data_dir' of the constructor where the new files should be placed.

And some constants that define the data to fetch:

- 'SOURCES': Any data sources defined by this constant will automatically be added to the queue of files to download.

Each source is a hash-reference with the following keys:

- 'FILE': The name of the file on the remote server
- 'SERVER': The path to the location of the file to fetch.

Further keys that can be defined include:

- 'POSTPROCESSOR': A code-reference which will called as a method and passed the downloaded file, and the location where it should end up.

1.5.2 Data Sources

Contents

Data Source Library

This page lists the current sources available for use in InterMine. All the sources here are found as ready-to-use JARs in the central repository, JCenter.

You can also add your own sources to load custom file formats, see *Writing your own data source* for more information. In addition, the *Tutorial* contains detailed steps on creating sources for a variety of different data formats.

Most of the configuration done in the config files is optional, if no config entry exists the default behaviour is followed. There are exceptions to this rule, however.

Core InterMine sources

These are commonly used sources that you may want to use to load data into your own InterMine instance.

Gene Ontology

GO Annotation Loads gene association files that link GO terms to genes or proteins.

Types of data loaded genes, proteins, GO terms, publications, GO evidence

How to download the data The data is available from http://www.geneontology.org

Configuration file (optional) There is an optional configuration file that let's you determine which type of object you create, and which identifier field you set. If your annotation file annotates genes and uses the primary identfier, these are the default values and you do not need to update the configuration file.

parameter	definition	possible values
typeAnnotated	class of what is being annotated	gene (default) or protein
identifier	which field to set	primaryIdentifier (default), symbol, or primaryAccession
readcolumn [4]	which column to use for identifier	identifier (default) or symbol

```
# an example entry
7165.typeAnnotated=protein
7165.identifier=primaryAccession
```

[4] See http://geneontology.org/docs/go-annotation-file-gaf-format-2.1/ for column descriptioins

How to load the data into your mine project XML example

```
<source name="go-annotation" type="go-annotation">
  <property name="src.data.dir" location="/data/go-annotation"/>
   <property name="ontologyPrefix" value="GO"/>
</source>
```

GO OBO Load the Gene Ontology term ids, names and definitions, and the relationships between terms. Should be loaded if the go-annotation source is used.

Types of data loaded GO terms

How to download the data From http://www.geneontology.org

How to load the data into your mine project XML example

```
<source name="go" type="go">
  <property name="src.data.file" location="/data/go-annotation/go-basic.obo"/>
</source>
```

go-basic.obo should load in a few minutes. *go.obo* is much more complex and takes a few hours and lots of memory.

Optional parameter: <property name="ontologyPrefix" value="FBbt"/>

This parameter causes the data parser to only load ontology terms with that prefix. Some OBO files have cross references that include ontology terms from other ontologies. Unfortunately the file doesn't include which terms correspond to which ontologies so we have to set the prefix.

Optional parameter: <property name="licence" value="https://creativecommons.org/licenses/by/4.0/"/>

This parameter will update the DataSet.licence field with the value you specify.

Homologue Data Sources InterMine comes with several data converter for homologue data, e.g. TreeFam, PANTHER, OrthoDB, Homlogene, etc. Follow the instructions below to include these datasets in your InterMine.

Treefam

Data ftp://ftp.sanger.ac.uk/pub/treefam/release-7.0/MySQL

Download two tables:

- *genes.txt.table*
- *ortholog.txt.table*

Project XML

```
<source name="treefam" type="treefam">
  <property name="src.data.dir" location="/DATA/treefam"/>
  <property name="src.data.dir.includes" value="ortholog.txt.table"/>
  <property name="geneFile" value="/DATA/treefam/genes.txt.table"/>
  <property name="treefam.organisms" value="7227 6239 7165 4932"/>
  <property name="treefam.homologues" value="9606 10090 10116 7955"/>
</source>
```

- ''treefam.organisms'' - all genes from the listed organisms will be processed
- ''treefam.homologues'' (optional) - genes will *only* be loaded into the database if they are a homologue of an organism of interest

Homologene

1. Data

ftp://ftp.ncbi.nih.gov/pub/HomoloGene/current/homologene.data

2. project.xml

```
<source name="homologene" type="homologene">
  <property name="src.data.dir" location="/DATA/homologene"/>
  <property name="homologene.organisms" value="7227 9606 10090 10116 7955 6239 4932"/>
</source>
```

OrthoDB Data

ftp://cegg.unige.ch/OrthoDB6/OrthoDB6_ALL_FUNGI_tabtext.gz, ftp://cegg.unige.ch/OrthoDB6/OrthoDB6_ALL_METAZOA_tabtex

Unzip the files and put them in the same directory.

Project XML

```
<source name="orthodb" type="orthodb">
  <property name="src.data.dir" location="/DATA/orthodb"/>
  <property name="orthodb.organisms" value="7227 9606 10090 10116 7955 6239 4932"/>
</source>
```

Panther Data

ftp://ftp.pantherdb.org/ortholog/current_release/RefGenomeOrthologs.tar.gz

gunzip to RefGenomeOrthologs.txt

Project XML

```
<source name="panther" type="panther">
  <property name="src.data.dir" location="/DATA/panther"/>
  <property name="panther.organisms" value="7227"/>
  <property name="panther.homologues" value="9606 10090 10116 7955 6239 4932"/>
</source>
```

Ensembl Compara

Download data from BioMart

1. [http://www.ensembl.org/biomart/martview/]
2. select database for primary organism, eg. *Ensembl Genes*
3. select dataset for primary organism, eg. *Drosophila melanogaster features (BDGP5.25)*
4. select FILTERS
   1. click on "FILTERS" on the left panel in BioMart (this will populate the main panel with filter options)

   2. select *MULTI SPECIES COMPARISONS*
   3. check the checkbox next to *Homolog filters*
   4. select the organism of interest in the dropdown
   1. eg. *Orthologous Caenorhabditis elegans Genes*
   2. make sure that next to the dropdown, *Only* is checked
1. select ATTRIBUTES
1. check the *Homologs* radio button at the top of the center panel
2. uncheck the *Ensembl Transcript ID* option, *Ensembl Gene ID* is now the only output
3. click on *ORTHOLOGS (Max select 6 orthologs):* to open that section of the form
4. select on the Gene ID for the organism of interest, eg. Drosophila Ensembl Gene ID
1. Run query
1. select the *[Results]* button at the top of the page
2. create *TSV* file, check box next to *Unique results only*
3. when prompted, save file as *TAXONID1_TAXONID2*

Add entry to project XML file

```
<source name="ensembl-compara" type="ensembl-compara">
  <property name="src.data.dir" location="/DATA/ensembl/compara"/>
  <property name="ensemblcompara.organisms" value="7227"/>
  <property name="ensemblcompara.homologues" value="6239"/>
</source>
```

Run build

Data file Tab-delimited files should be named <TAXON ID>_<TAXON ID>, eg. 9606_10090 for a file with human genes and mouse orthologues.

Gene ID	Homologue ID
ENSG00000253023	ENSMUSG00000088328
ENSG00000238364	ENSMUSG00000088728

Download script When you have created your query, you can export the Perl script or XML so you can run the query automatically next time, eg:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Query>
<Query  virtualSchemaName = "default" formatter = "TSV" header = "0" uniqueRows = "0" count = "" data

  <Dataset name = "hsapiens_gene_ensembl" interface = "default" >
    <Filter name = "with_dmelanogaster_homolog" excluded = "0"/>
    <Attribute name = "ensembl_gene_id" />
    <Attribute name = "drosophila_ensembl_gene" />
  </Dataset>
</Query>
```

Identifiers The default rule for bio-InterMine is to put the MOD identifiers (eg. MGI:XXX or ZDB-GENE-XXX) in the primaryIdentifier field. This is tricky because some homologue sources use the Ensembl identifiers (Ensembl identifiers belong in the Gene.crossReferences collection).

To solve this problem, each homologue source uses the NCBI identifier resolver. This resolver takes the Ensembl ID and replaces it with the corresponding MOD identifier.

How to use an ID resolver

1. Download the identifier file - ftp://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz
2. Unzip the file to */DATA_DIR/ncbi/gene_info*

Warning: Make sure permissions on the file are correct so the build process can read this file.

3. Download the identifier file for humans - http://www.flymine.org/download/idresolver/humangene to another directory, eg. /DATA_DIR/human/identifiers
4. Create a sub directory */DATA_DIR/idresolver/* as file root path and add symbolic links to the two files.

```
$ cd /DATA_DIR/idresolver/
$ ln -s /DATA_DIR/ncbi/gene_info entrez
$ ln -s /DATA_DIR/human/identifiers humangene
```

5. Add the root path to the file in *~/.intermine/MINE.properties*

```
resolver.file.rootpath=/DATA_DIR/idresolver/
```

See *Id Resolvers* for details on how ID resolvers work in InterMine.

Warning: The entrez identifiers file appears to only have the sequence identifier for worm instead of the WBgene identifier

Alternately you can load identifier sources.

Here are the download scripts we use here at InterMine:

Data Download

We use WormMart but are happy to hear of a better source for worm identifiers.

Here are the project XML entries used by FlyMine:

FlyMine Project XML

Interactions

BioGRID Loads interactions data from BioGRID

Types of data loaded genes, proteins, interactions

How to download the data From the download section of the BioGRID website: http://thebiogrid.org

Download the file named: *BIOGRID-ORGANISM-[version].psi25.zip*.

How to load the data into your mine

project XML example

```
<source name="biogrid" type="biogrid">
  <property name="src.data.dir" location="/DATA/biogrid"/>
  <property name="src.data.dir.includes" value="*psi25.xml"/>
  <property name="biogrid.organisms" value="7227 6239 4932"/>
</source>
```

biogrid_config.properties Determines which gene identifiers are set. organisms - If none are configured, all interactions are stored.

This is what the gene looks like in biogrid

```
 <names>
         <shortLabel>CG1111</shortLabel>
 </names>
 <xref>
<primaryRef db="FLYBASE" id="FBgn001" />
```

shortLabel

To set your gene.identifier to be the shortLabel in the biogrid XML, use this config:

```
<TAXON_ID>.<GENE_IDENTIFIER_FIELD>=shortLabel
```

xref

To set your gene.identifier field to be a value from an xref entry, use this syntax:

```
<TAXON_ID>.xref.<GENE_IDENTIFIER_FIELD> = <XREF_DB_VALUE>
```

Note: xref "db" value is not case sensitive, case seems to vary from file to file.

IntAct Loads binary interactions data from IntAct

Types of data loaded genes, interactions

How to download the data ftp://ftp.ebi.ac.uk/pub/databases/IntAct/current/psi25/species/

How to load the data into your mine

project XML example

```
<source name="psi-intact" type="psi" dump="true">
  <property name="src.data.dir" location="/data/intact"/>
  <property name="intact.organisms" value="7227"/>
</source>
```

psi-intact_config.properties Determines which gene identifiers are set. organisms - If none are configured, all interactions are stored.

IntAct - complexes Loads complex interaction data from IntAct

Types of data loaded genes, interactions, complexes, publications

How to download the data ftp://ftp.ebi.ac.uk/pub/databases/intact/complex/current/psi25/

How to load the data into your mine

project XML example

```
<source name="psi-complexes" type="psi-complexes">
  <property name="src.data.dir" location="/DATA/psi/intact/complexes/current"/>
  <property name="complexes.source" value="sgd"/>
</source>
```

There is also a corresponding displayer for these data.

PSI-MI Ontology Include this source when loading *psi* data to fill in details of ontology terms used. Should be loaded if you are loading interaction data.

Types of data loaded ontology terms

How to download the data https://raw.githubusercontent.com/HUPO-PSI/psi-mi-CV/master/psi-mi.obo

How to load the data into your mine project XML example

```
<source name="psi-mi-ontology" type="psi-mi-ontology">
  <property name="src.data.file" location="/data/psi/psi-mi.obo"/>
</source>
```

Pathway data sources Content:

KEGG Link genes to KEGG pathways that they operate in.

Types of data loaded genes, pathways

How to download the data http://www.genome.jp/kegg

How to load the data into your mine

project XML example

```
<source name="kegg-pathway" type="kegg-pathway">
  <property name="src.data.dir" location="/data/kegg"/>
  <property name="kegg.organisms" value="7227"/>
</source>
```

kegg_config.properties Decides which gene identifier fields are populated, mapping from organism taxonId to abbreviation. Only taxonIds specified in project.xml file are downloaded, if no taxonIds are configured, all are loaded. For example:

```
# bacteria
eco.taxonId = 511145
```

Reactome

Types of data loaded proteins, genes, pathways

How to download the data http://www.reactome.org/download/current/UniProt2Reactome_All_Levels.txt

How to load the data into your mine project XML example

```
<source name="reactome" type="reactome">
  <property name="src.data.dir" location="/data/reactome" />
  <property name="reactome.organisms" value="9606 10090" />
</source>
```

This source contains a task to copy the Pathways from the proteins to the related genes. To include this, add this to the *post-processing* section of your project XML file:

```
<post-processing>
  <post-process name="do-sources" />
  ...
</post-processing>
```

See *Post processing* for more information on post-processing.

Proteins

UniProt

Types of data loaded genes, proteins, GO annotation, protein domains, publications, UniProt features, comments, synonyms, cross references, EC numbers, components

How to download the data This source loads data from the UniProt website here: ftp://ftp.uniprot.org/pub/databases/uniprot/current_release

The UniProt source expects the data files to be in a special format:

```
TAXONID_uniprot_sprot.xml
TAXONID_uniprot_trembl.xml
```

To download a single taxon, you can use this URL:

http://www.uniprot.org/uniprot/?format=xml&query=taxonomy%3A9606+AND+reviewed%3Ayes&compress=yes

parameter	value
taxonomy	e.g. 9606 for human
reviewed	yes for swiss prot, no for trembl
compress	if yes, zipped

How to load the data into your mine

Configuration

Gene identifier fields You can specify which gene fields are assigned when UniProt data is loaded. An example entry:

```
10116.uniqueField = primaryIdentifier
10116.primaryIdentifier.dbref = RGD
10116.secondaryIdentifier.dbref = Ensembl
10116.symbol.name = primary
```

The format for the file is:

<TAXON_ID>.<IDENTIFIER_FIELD> = <VALUE>

An example A rat uniprot entry: http://www.uniprot.org/uniprot/Q923K9.xml

The second line of that configuration would set the ID value as the gene.primaryIdentifier:

```
<dbReference type="RGD" id="619834" key="33">
        <property type="gene designation" value="Acf"/>
</dbReference>
```

The third line would set this ID value as gene.secondaryIdentifier:

```
<dbReference type="Ensembl" id="ENSRNOG00000033195" key="30">
        <property type="organism name" value="Rattus norvegicus"/>
</dbReference>
```

The last line would set the value between the <name/> tags as gene.symbol:

```
<gene>
        <name type="primary">A1cf</name>
        <name type="synonym">Acf</name>
        <name type="synonym">Asp</name>
</gene>
```

The values for symbol.name can be primary, ORF or ordered locus.

Protein feature types You can also configure which protein features to load.

To load specific feature types only, specify them like so:

```
# in uniprot_config.properties
feature.types = helix
```

To load NO feature types:

```
# in uniprot_config.properties
feature.types = NONE
```

To load ALL feature types, do not specify any feature types, remove that line from this config file. Loading all feature types is the default behaviour.

Project.xml

```
<source name="uniprot" type="uniprot" >
  <property name="uniprot.organisms" value="7227 9606"/>
  <property name="src.data.dir" location="/data/uniprot"/>
  <property name="creatego" value="true"/>
  <property name="creategenes" value="true"/>
  <property name="allowduplicates" value="false"/>
  <property name="loadfragments" value="false"/>
  <property name="loadtrembl" value="true"/>
</source>
```

property	description	default
creategenes	if TRUE, process genes	true
creatego	if TRUE, process GO annotation	false
allowduplicates	if TRUE, allow proteins with duplicate sequences to be processed	false
loadfragments	if TRUE, load all proteins even if isFragment = true	false
loadtrembl	if FALSE, not load trembl data for given organisms, load sprot data only	true

FASTA This source loads FASTA data for isoforms. The UniProt entry is does not contain the sequences for isoforms.

ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/uniprot_sprot_varsplic.fasta.gz

```
<source name="uniprot-fasta" type="fasta">
  <property name="fasta.taxonId" value="7227 9606"/>
  <property name="fasta.className" value="org.intermine.model.bio.Protein"/>
  <property name="fasta.classAttribute" value="primaryAccession"/>
  <property name="fasta.dataSetTitle" value="UniProt data set"/>
  <property name="fasta.dataSourceName" value="UniProt"/>
  <property name="src.data.dir" location="/data/uniprot/current"/>
  <property name="fasta.includes" value="uniprot_sprot_varsplic.fasta"/>
  <property name="fasta.sequenceType" value="protein" />
  <property name="fasta.loaderClassName" value="org.intermine.bio.dataconversion.UniProtFastaLoaderTa
</source>
```

UniProt keywords Loads the names for the UniProt keywords contained in the main UniProt converter.

ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/docs

```
<source name="uniprot-keywords" type="uniprot-keywords">
  <property name="src.data.dir" location="/data/uniprot/current"/>
  <property name="src.data.dir.includes" value="keywlist.xml"/>
</source>
```

InterPro InterMine has two InterPro data sources. One that loads the protein domains, e.g. name and description and one that loads the relationship between the proteins and domains.

Types of data loaded protein domains, e.g. name and description

How to download the data ftp://ftp.ebi.ac.uk/pub/databases/interpro/interpro.xml.gz

How to load the data into your mine project XML example

```
<source name="interpro" type="interpro">
  <property name="src.data.dir" location="/data/interpro"/>
</source>
```

InterPro to protein This source queries for proteins already in the database and loads related protein domains. So this source must be run after UniProt.

Types of data loaded protein domains, their relationship to the protein and protein domain region

How to download the data ftp://ftp.ebi.ac.uk/pub/databases/interpro/protein2ipr.dat.gz ftp://ftp.ebi.ac.uk/pub/databases/interpro/match_complete.dat.gz

How to load the data into your mine project XML example

```
<!-- has to be after UniProt because only loads protein domains for loaded proteins -->
<source name="protein2ipr" type="protein2ipr">
    <property name="src.data.dir" location="/data/interpro"/>
    <property name="src.data.dir.includes" value="protein2ipr.dat"/>
    <property name="protein2ipr.organisms" value="9606"/>
    <property name=”osAlias” value=”os.production”/>
</source>
```

Publications

PubMed Data from pubmed. entire file is downloaded, only taxon IDs set in project.xml will be loaded. if nothing configured, processes all entries.

Types of data loaded genes, publications

How to download the data files

- ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene2pubmed.gz
- ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene_info.gz

How to load the data into your mine project XML example

```
<source name="pubmed-gene" type="pubmed-gene">
  <property name="src.data.dir" location="DATA_DIR/pubmed/" />
  <property name="pubmed.organisms" value="7227"/>
  <property name="src.data.dir.includes" value="gene2pubmed"/>
</source>
```

Publications All publications are referred to by PubMed id by other sources. This source should be included at the end of the build. It will query all PubMed ids from the database (where the *title*, *year*, or *first author* columns are NULL), fetch details from the Entrez web service and fill in Publication objects.

Types of data loaded None, the publciation records already in the database are updated.

How to download the data Data is fetched from the NCBI web site for publication records already in the InterMine database.

How to load the data into your mine project XML example

```
<source name="update-publications" type="update-publications" dump="true">
  <property name="src.data.file" location="publications.xml"/>
  <!-- <property name="loadFullRecord" value="true"/> -->
</source>
```

properties:

1. loadFullRecord - load MeSH terms and abstract, value "true"/"false"

NCBI - Entrez gene Gene information from NCBI

Types of data loaded genes

How to download the data files

- ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/GENE_INFO/All_Data.gene_info.gz

Be sure to unzip the file.

How to load the data into your mine project XML example

```
<source name="ncbi-gene" type="ncbi-gene">
  <property name="src.data.dir" location="/DATA_DIR/ncbi" />
  <property name="organisms" value="9606" />
</source>
```

Chado We have developed an InterMine data source that can use a GMOD Chado database as a source for an InterMine warehouse. The eventual aim is to allow import of any Chado database with some configuration. This will provide a web environment to perform rapid, complex queries on Chado databases with minimal development effort.

Converter The converter for this source is the *ChadoDBConverter* class. This class controls which *ChadoProcessors* are run. A *ChadoProcessor* class corresponds to a chado module. For example, the sequence module is processed by the *SequenceProcessor* and the stock module is processed by the *StockProcessor*.

Chado tables The *chado-db* source is able to integrate objects from a Chado database. Currently only tables from the *Chado sequence module* and *Chado stock modules* are read.

These tables are queried from the chado database:

feature used to create objects in the ObjectStore

- The default configuration only supports features that have a Sequence Ontology type (eg. *gene*, *exon*, *chromosome*)
- Each new feature in InterMine will be a sub-class of *SequenceFeature*.

featureloc used to create *Location* objects to set *chromosomeLocation* reference in each *SequenceFeature*

feature_relationship used to find *part_of* relationships between features

- this information is used to create parent-child references and collections
- examples include setting the *transcripts* collection in the *Exon* objects and the *gene* reference in the *Transcript* class.

dbxref* and *feature_dbxref used to create *Synonym* objects for external identifiers of features

- the *Synonym's will be added to the 'synonyms* collection of the relevant *SequenceFeature*

featureprop used to set fields in features based on properties

- an example from the FlyBase database: the *SequenceFeature.cytoLocation* field is set using the *cyto_range* feature_prop

synonym* and *feature_synonym used to create extra *Synonym* objects for *chado* synonyms and to set fields in features

- the *Synonym's will be added to the 'synonyms* collection of the relevant *SequenceFeature*

cvterm* and *feature_cvterm used to set fields in features and to create synonyms based on CV terms

pub*, *feature_pub* and *db used to set the *publications* collection in the new *SequenceFeature* objects.

Additionally, the *StockProcessor* class reads the tables from the chado stock module, eg. stockcollection, stock, stock_genotype.

Default configuration The default configuration of *ChadoDBConverter* is to query the *feature* table to only a limited list of types. The list can be changed by sub-classing the *ChadoDBConverter* class and overriding the *getFeatureList()* method. The *featureloc*, *feature_relationship* and *pub* tables will then be queried to create locations, parent-child relationships and publications (respectively).

Converter configuration Sub-classes can control how the Chado tables are used by overriding the *getConfig()* method and returning a configuration map.

Source configuration Example source configuration for reading from the ''C.elegans" Chado database:

```
<source name="chado-db-wormbase-c_elegans" type="chado-db" dump="true">
  <property name="source.db.name" value="wormbase"/>
  <property name="genus" value="Caenorhabditis"/>
  <property name="species" value="elegans"/>
  <property name="taxonId" value="6239"/>
  <property name="dataSourceName" value="WormBase"/>
  <property name="dataSetTitle" value="WormBase C.elegans data set"/>
</source>
```

Sub-classing the converter The processor classes can be sub-classed to allow organism or database specific configuration. To do that, create your class (perhaps in *bio/sources/chado-db/main/src/*) set the *processors* property in your source element. For example for reading the FlyBase Chado database there is a *FlyBaseProcessor* which can be configured like this:

```
<source name="chado-db-flybase-dmel" type="chado-db">
...
        <property name="processors" value="org.intermine.bio.dataconversion.FlyBaseProcessor"/>
...
```

Current uses FlyMine uses the *chado-db* source for reading the ''Drosophila" genomes from the FlyBase *chado* database. The *FlyBaseProcessor* sub-class is used for configuration and to handle FlyBase special cases.

modMine for the modENCODE project uses *ChadoDBSource* for reading ''D. melanogaster" from FlyBase and to read ''C. elegans" data from the WormBase *chado* database. The *WormBaseProcessor* sub-class is used for configuration when reading from WormBase.

Implementation notes for the chado-db source The *chado-db* source is implemented by the *ChadoDBConverter* class which runs the *ChadoProcessor's that have been configured in the 'project.xml*. The configuration looks like this:

```
<source name="chado-db-some-database" type="chado-db">
  ...
  <property name="processors" value="org.intermine.bio.dataconversion.ChadoSequenceProcessor org.inte
  ...
```

ChadoDBConverter.process() will create an object for each *ChadoProcessor* in turn, then call *ChadoProcessor.process()*.

Chado sequence module table processing *ChadoSequenceProcessor* processes the sequence module from Chado. The *process()* method handles each table in turn by calling: *processFeatureTable()*, *processFeatureCVTermTable()* etc.

Each table processing method calls a result set method, eg. *processFeatureTable()* calls *getFeatureTableResultSet()* and then processes each row. The returned *ResultSet* may not always include all rows from the Chado table. For example the *getFeatures()* method returns a sub-set of the possible feature types and that list is used to when querying the feature table.

Generally each row is made into an appropriate object, eg. in *processFeatureTable()*, *feature* table rows correspond to *BioEntity* objects. Some rows of some tables are ignored (ie. not turned into objects) based on configuration.

Reading the feature table Handled by *ChadoSequenceProcessor.processFeatureTable()*

For each feature it calls: *ChadoSequenceProcessor.makeFeatureData()*, which may be overridden by subclasses. If *makeFeatureData()* returns null (eg. because the sub-class does not need that feature) the row is discarded, otherwise processing of the feature continues.

Based on the configuration, fields in the *BioEntity* are set using *feature.uniquename* and *feature.name* from Chado.

If the *residues* column in the feature is set, create a *Sequence* object and add it to the new *BioEntity*.

Reading the featureloc table Handled by *ChadoSequenceProcessor.processLocationTable()*.

This method gets passed a result set with start position, end position and information from the *featureloc* table. For each row from the result set it will:

- store a *Location* object
- set *chromosomeLocation* in the associated *SequenceFeature*
- set the *chromosome* reference in the *SequenceFeature* if the *srcfeature* from the *featureloc* table is a chromosome feature

Reading the feature_relationship table Handled by *ChadoSequenceProcessor.processRelationTable()*.

This method calls *getFeatureRelationshipResultSet()* to return the relations of interest. The relations will be used to create references and collections.

The method will automatically attempt to find and set the appropriate references and collections for *part_of* relations. As an example, if there is a *part_of* relation in the table between *Gene* and *Transcript* and there *Gene.transcript* reference or a *Gene.transcripts* collection, it will be set.

There are two modes of operation, controlled by the *subjectFirst* parameters. If true, order by the *subject_id* of the *feature_relationship* table so we get results like:

gene1_feature_id	relation_type	protein1_feature_id
gene1_feature_id	relation_type	protein2_feature_id
gene2_feature_id	relation_type	protein1_feature_id
gene2_feature_id	relation_type	protein2_feature_id

(Assuming the unlikely case where two genes are related to two proteins)

If *subjectFirst* is false we get results like:

gene1_feature_id	relation_type	protein1_feature_id
gene2_feature_id	relation_type	protein1_feature_id
gene1_feature_id	relation_type	protein2_feature_id
gene2_feature_id	relation_type	protein2_feature_id

The first case is used when we need to set a collection in the gene, the second if we need to set a collection in proteins.

Reading the cvterm table Handled by *ChadoSequenceProcessor.processFeatureCVTermTable()*

Using the default chado source

1. Add the chado database to your MINE_NAME.properties file, eg:

```
db.flybase.datasource.class=org.postgresql.ds.PGPoolingDataSource
db.flybase.datasource.dataSourceName=db.flybase
db.flybase.datasource.serverName=SERVER_NAME
db.flybase.datasource.databaseName=DATABASE_NAME
db.flybase.datasource.user=USER_NAME
db.flybase.datasource.password=SECRET_PASSWORD
db.flybase.datasource.maxConnections=10
db.flybase.driver=org.postgresql.Driver
db.flybase.platform=PostgreSQL
```

The chado database has to be on the local network.

2. Add source to project XML file

```
<source name="chado-db" type="chado-db">
  <property name="source.db.name" value="flybase"/>
  <property name="organisms" value="7227"/>
  <property name="dataSourceName" value="FlyBase"/>
  <property name="converter.class" value="org.intermine.bio.dataconversion.ChadoDBConverter"/>
  <property name="processors" value="org.intermine.bio.dataconversion.SequenceProcessor"/>
</source>
```

3. Run the build

```
flymine $ ./gradlew clean builddb
flymine $ ./gradlew integrate -Psource=chado-db
```

See *Database Building* for more information on running builds.

This will load the data using the default chado loader. If you want to load more data you will have to write a custom chado converter. FlyMine uses a FlyBase chado "processor" to parse interactions, etc. See FlyBaseProcessor.java for an example.

Tripal The Chado specific tables are not in the postgres default "public" schema of the database. Instead, Tripal puts it in a postgres schema named "chado".

To workaround this, you would need to alter your Chado processor to run this query first, before running any SELECT statements:

```
ALTER DATABASE <dbname> SET search_path TO chado, public
```

Starting with **InterMine 1.8**, you can instead directly define the schema in the properties of the database in your properties file, like

```
db.your_source.datasource.schema=your_schema
```

for example

```
db.tripaldbname.datasource.schema=chado
```

FASTA

Types of data loaded features and their sequences. Will create a feature for each entry in a fasta file and set the sequence, the class of the feature to create is set for the whole file.

How to download the data N/A - will parse any file in FASTA format

How to load the data into your mine project XML example

```
<source name="flybase-dmel-gene-fasta" type="flybase-dmel-gene-fasta">
  <property name="flybase-dmel-gene-fasta.taxonId" value="7227"/>
  <property name="flybase-dmel-gene-fasta.dataSetTitle" value="FlyBase fasta data set for Drosophila
  <property name="flybase-dmel-gene-fasta.dataSourceName" value="FlyBase"/>
  <property name="flybase-dmel-gene-fasta.className" value="org.intermine.model.bio.Gene"/>
  <property name="flybase-dmel-gene-fasta.classAttribute" value="primaryIdentifier"/>
  <property name="flybase-dmel-gene-fasta.includes" value="dmel-all-gene-*.fasta"/>
  <property name="src.data.dir" location="/DATA/flybase/fasta"/>
  <!-- add licence here -->
  <property name="flybase-dmel-gene-fasta.licence" value="https://creativecommons.org/licenses/by/4.
</source>
```

attribute	content	purpose
taxonId	space-delimited list of taxonIds	only features with the listed taxonIds will be loaded
className	fully-qualified class name	determines which feature will be loaded
classAttribute	identifier field from className	determines which field from the feature will be set
dataSetTitle	name of dataset	determines name of dataset object
dataSource-Name	name of datasource	determines name of datasource object
src.data.dir	location of the fasta data file	these data will be loaded into the database
includes	name of data file	this data file will be loaded into the database
sequenceType	class name	type of sequence to be loaded
loaderClass-Name	name of Java file that will process the fasta files	only use if you have created a custom fasta loader
licence	URL pointing to standard data licence for data	updates DataSet.licence with value

GFF3 InterMine comes with a GFF parser which loads GFF3 data files into your mine - without writing any Perl or Java code. This isn't a source itself but genome annotation from gff files can be loaded easily by creating a new source of type gff. See redfly, malaria-gff and tiffin for examples.

Configuration is added to the *project.properties* file and an optional handler can be added to deal with data in the attributes section of the gff file.

Types of data loaded sequence features

How to download the data N/A - will parse any file in GFF3 format

How to load the data into your mine

1. place valid GFF3 files into a directory
2. add entry to project XML file
3. run build

```
# example GFF3 file
MAL1    ApiDB   gene    183057  184457  .       -       .       ID=gene.46311;description=hypothetica
MAL1    ApiDB   mRNA    183057  184457  .       +       .       ID=mRNA.46312;Parent=gene.46311
```

If you follow the above steps with this data file, the following will happen:

1. gene and mRNA objects created
2. "MAL1" will be the identifier
3. start = 183057, end = 184457
4. gene will be located in -1 strand, mRNA will be located on the 1 strand.

Configuration File By default, columns such as "type", "start", "end", "strand" and "ID" field in "attributes" column are parsed automatically. To do more processing or access the attributes, you are able to configure in *gff_config.properties*. This file should live in your mine's *dbmodel/resources* directory.

```
# gff_config.properties example for E. coil gff3 attributes
511145.terms=gene,exon                                  # feature types to load, e.g. load gene and exon
511145.excludes=CDS                                     # comma-separated list of feature types to exclude
511145.gene.attributes.Dbxref.EcoGene=primaryIdentifier # use Dbxref EcoGene field as primaryIdentifi
511145.gene.attributes.locus_tag=secondaryIdentifier    # use locus_tag field as secondaryIdentifier
511145.attributes.gene=symbol                           # use gene field as symbol
511145.attributes.gene_synonym=synonym                  # use gene_synonym field for synonym
511145.exon.attributes.type=scoreType                   # a class specific attribute
```

For more advanced processing, you will have to write your own GFF3 parser.

Parent relationship The parent-child relationship between features can also be handled automatically if you set it up properly. Take MalariaGFF3RecordHandler for example:

```
public MalariaGFF3RecordHandler(Model tgtModel) {
    super(tgtModel);
    // refsAndCollections controls references and collections that are set from the
    // Parent= attributes in the GFF3 file.
    refsAndCollections.put("Exon", "transcripts");
    refsAndCollections.put("MRNA", "gene");
}
```

Project XML Here is an example GFF3 entry in the project XML file:

```
    # add to project.xml file
    # NOTE: update the "type" if you are using your own custom GFF3 parser

<source name="example-gff3" type="gff">
  <property name="gff3.taxonId" value="9606"/>
  <property name="gff3.seqClsName" value="Chromosome"/>
  <property name="src.data.dir" location="/DATA/*.gff3"/>
  <property name="gff3.dataSourceName" value="NCBI"/>
  <property name="gff3.dataSetTitle" value="Release GRCh38 of the Homo sapiens genome sequence"/>
  <!-- add licence here -->
  <property name="gff3.licence" value="https://creativecommons.org/licenses/by-sa/3.0/" />
</source>
```

Here are the descriptions of the properties available:

property	example definition	
gff3.seqClsName	Chromosome	the ids in the first column represent Chromosome objects, e.g. MAL1
gff3.taxonId	36329	taxon id
gff3.dataSourceName	PlasmoDB	the data source for features and their identifiers, this is used for the DataSet (evidence) and synonyms.
gff3.seqDataSourceName	PlasmoDB	the source of the seqids (chromosomes) is sometimes different to the features described
gff3.dataSetTitle	PlasmoDB P. falciparum genome	a DataSet object is created as evidence for the features, it is linked to a DataSource (PlasmoDB)
gff3.licence	https://creativecommons.org/licenses/by-sa/3.0/	URL to a standard data licence

Writing a custom GFF parser You can extend the generic parser by writing your own Java code to process the GFF3 data.

Make Source script Create your custom source by running the create source script:

```
$ ./bio/scripts/make_source mouse-cdna gff
created /home/USER_NAME/git/bio/sources/mouse-cdna directory for mouse-cdna
```

The script has created a new source for you in the *bio/sources* directory.

Java code The Java file you now want to edit is here: *bio/sources/SOURCE_NAME/main/src/org/intermine/bio/dataconversion*

The *process()* method is called for every line of GFF3 file(s) being read. Features and their locations are already created but not stored so you can make changes here. Attributes are from the last column of the file are available in a map with the attribute name as the key. For example:

```
Item feature = getFeature();
String symbol = record.getAttributes().get("symbol");
feature.setAttribute("symbol", symbol);
```

Any new Items created can be stored by calling addItem(). For example:

```
String geneIdentifier = record.getAttributes().get("gene");
gene = createItem("Gene");
gene.setAttribute("primaryIdentifier", geneIdentifier);
addItem(gene);
```

You should make sure that new Items you create are unique, i.e. by storing in a map by some identifier.

It may be helpful to look at current GFF3 parsers:

1. *LongOligoGFF3RecordHandler.java*
2. *MirandaGFF3RecordHandler.java*
3. *RedFlyGFF3RecordHandler.java*
4. *FlyRegGFF3RecordHandler.java*
5. *DrosDelGFF3RecordHandler.java*

See *Tutorial* for more information on how to run a GFF source.

Identifier Data Sources You can load MODs ids into your mine using identifier data sources.

Types of data loaded genes

How to download the data

flybase-identifiers http://flybase.org/static_pages/downloads/FB20XX_XX/synonyms/fb_synonym_fb_FB20XX_XX.tsv.gz
- where FB20XX_XX = the current FlyBase release

zfin-identifiers http://zfin.org/downloads/ensembl_1_to_1.txt

sgd-identifiers http://downloads.yeastgenome.org/curation/chromosomal_feature/SGD_features.tab

wormbase-identifiers query wormbase biomart webservice

mgi-identifiers ftp://ftp.informatics.jax.org/pub/reports/MGI_Coordinate.rpt

rgd-identifiers ftp://rgd.mcw.edu/pub/data_release/GENES_RAT.txt

How to load the data into your mine project XML example

```
 <source name="flybase-identifiers" type="flybase-identifiers">
   <property name="src.data.dir" location="/DATA/flybase-identifiers"/>
 </source>

 <source name="zfin-identifiers" type="zfin-identifiers">
   <property name="src.data.dir" location="/DATA/zfin-identifiers"/>
 </source>

 <source name="sgd-identifiers" type="sgd-identifiers">
   <property name="src.data.dir" location="/DATA/sgd-identifiers"/>
 </source>

 <source name="wormbase-identifiers" type="wormbase-identifiers">
   <property name="src.data.dir" location="/DATA/worm-identifiers"/>
 </source>

<source name="mgi-identifiers" type="mgi-identifiers">
   <property name="src.data.dir" location="/DATA/mgi-identifiers"/>
 </source>

<source name="rgd-identifiers" type="rgd-identifiers">
   <property name="src.data.dir" location="/DATA/rgd-identifiers"/>
 </source>
```

InterMine Items XML Use this source to load *InterMine Items XML* conforming to the data model directly into the production database.

intermine-items-xml-file Use this source to load *InterMine Items XML* conforming to the data model directly into the production database.

intermine-items-large-xml-file Use this source to load *InterMine Items XML* conforming to the data model into the production database, this uses an intermediate database to allow it to cope with very large files that would otherwise cause memory problems.

Types of data loaded Any

How to load the data into your mine See *Writing your own data source* for details on how to do this.

project XML example

```
<source name="arbeitman-items-xml" type="arbeitman-items-xml">
  <property name="src.data.file" location="/data/arbeitman/arbeitman-tgt-items.xml"/>
</source>
```

OMIM

Types of data loaded genes, diseases

How to download the data Contact OMIM for your API key. Use our script to download the data.

How to load the data into your mine project XML example

```
<source name="omim" type="omim">
  <property name="src.data.dir" location="/data/omim"/>
</source>
```

Organisms All other sources refer to organisms only by their NCBI taxonomy id. This source should be included at the end of the build. It will select the taxonIds loaded into the Organism class, fetch details via the Entrez web service and fill in the organism names in the database.

Types of data loaded update organism entries

How to download the data N/A - source uses NCBI's web services

How to load the data into your mine project XML example

```
<source name="entrez-organism" type="entrez-organism">
  <property name="src.data.file" location="build/organisms.xml"/>
</source>
```

Sequence Ontology (SO) This source loads no data but adds a class in the data model for every term in the sequence ontology in your data model. SO terms represent biological features such as gene, exon, 3' UTR. You should include this source if you are loading genome annotation.

Types of data loaded Sequence Ontology terms

How to download the data Included in InterMine source code

How to load the data into your mine project XML example

```
<source name="so" type="so">
  <property name=”src.data.file” location=”so.obo” />
  <property name="licence" value="https://creativecommons.org/licenses/by/4.0/"/>
</source>
```

To add or remove SO terms from your model, update your *so_terms* file in *dbmodel/resources*

Uberon

Types of data loaded ontology terms

How to download the data http://purl.obolibrary.org/obo/uberon.obo

How to load the data into your mine project XML example

```
<source name="uberon" type="uberon">
  <property name="src.data.file" location="/data/uberon/uberon.obo"/>
</source>
```

Data Sources Load the official title, description and URL for data sources.

Types of data loaded Update data source entries

How to download the data http://www.uniprot.org/docs/dbxref.txt

How to load the data into your mine project XML example

```
<source name="update-data-sources" type="update-data-sources">
  <property name="src.data.file" location="datasources.xml"/>
  <property name="dataSourceFile" value="/data/uniprot/xrefs/dbxref.txt"/>
</source>
```

Data Sets Load an XML file with details of your data sets and associated information, e.g. description and URL

Types of data loaded Update data source and data set entries

How to download the data Create your own datasets.xml file with your data in InterMine items XML format and put in your mine’s *dbmodel/resources* directory so that it’s on your classpath.

```xml
<?xml version="1.0"?>
<items>
<item id="09" class="" implements="DataSource">
    <attribute name="name" value="NCBI"/>
    <attribute name="description" value="National Centre for Biotechnology Information"/>
    <attribute name="url" value="https://www.ncbi.nlm.nih.gov"/>
</item>
<item id="10" class="" implements="DataSet">
    <attribute name="name" value="Homo sapiens genome sequence"/>
    <attribute name="description" value="Release GRCh38 of the Homo sapiens genome sequence"/>
    <attribute name="version" value="GRCh38.p12"/>
    <attribute name="url" value="https://www.ncbi.nlm.nih.gov"/>
    <reference name="dataSource" ref_id="09"/>
</item>
</items>
```

How to load the data into your mine project XML example

```xml
<source name="flymine-static" type="flymine-static">
  <property name="src.data.file" location="/data/datasets.xml"/>
</source>
```

VCF files Load SNP data from a VCF file

Types of data loaded SNPs

How to download the data First you will need a VCF file, here is an example:

> ftp://ftp.ensembl.org/pub/release-79/variation/vcf/homo_sapiens/

How to load the data into your mine

Add vcf to the list of datasources to be integrated

```xml
<source name="my-data-source" type="vcf">
  <property name="src.data.dir" location="/data/variation/current" />
  <property name="vcf.includes" value="*.vcf" />
  <property name="vcf.vcfTaxonId" value="9606" />
  <property name="vcf.vcfDataSetTitle" value="Ensembl SNP data set" />
  <property name="vcf.vcfDataSourceName" value="Ensembl" />
</source>
```

FlyMine Specific sources

These are sources that load Drosophila specific data sets into FlyMine, we don't expect you will re-use these unless you are creating a Drosophila warehouse. All of these sources are located in https://github.com/intermine/flymine-bio-sources.

- affy-probes
- arbeitman-items-xml
- bdgp-clone

- bdgp-insitu
- drosdel-gff
- drosophila-homology
- fly-anatomy-ontology
- flyatlas
- flybase-alleles
- flybase-expression
- fly-development-ontology
- fly-fish
- fly-misc-cvterms
- flyreg
- long_oligo
- miranda
- redfly
- rnai

See FlyMine for more information about these datasets. Look at FlyMine's project.xml for examples of how to use these sources.

HumanMine Specific sources

- arrayexpress-atlas
- atlas-express
- clinvar
- ensembl-hgnc
- gtex
- hgnc
- hpo
- hpo-annotation
- huge-gwas
- human-gene
- mgi-alleles
- ncbi-summaries
- orphanet
- protein-atlas

See HumanMine for more information about these datasets. Look at HumanMine's project.xml for examples of how to use these sources.

Writing your own data source

The aim of this tutorial is to create a new data source to parse your data file and load the data into your InterMine database.

There are three parts to creating a new source:

1. Create a directory for your data sources, e.g. *flymine-bio-sources*
2. Write a data parser
3. Configure the mine to use this new source

To get started, create a directory to put all of your data sources in. You only need to that once. Then follow the instructions below and run the script to create your data source. If necessary, use the APIs provided to write code to parse your data file. Finally, add your new data source to your project XML file.

1. Create bio-sources directory

You only need to do this once, but you need a directory to hold all of your mine's data sources.

- Place next to your mine, e.g. ~/git/flymine and ~/git/flymine-bio-sources
- Keep in source control. Please. We use Git.

2. Run make_source script

Checkout the intermine scripts repository that contains the *make_source* script.

```
# check out the repository that has the scripts we need
~/git $ git clone https://github.com/intermine/intermine-scripts.git
```

The *make_source* script creates the basic skeleton for a source. It should be run in your mine's data sources directory, like this:

```
# run the script in the directory you created for your mine data sources
~/git/flymine-bio-sources $ ~/git/intermine-scripts/make_source $SOURCE_NAME $SOURCE_TYPE
```

SOURCE_NAME The name of your source, e.g. uniprot-fasta or biogrid. The script expects a lowercase word without any special characters (except dashes, dashes are fine).

SOURCE_TYPE The type of your source. One of six options, see below.

Which source type do I need? It depends! If you want to use Java and have a custom data file, use *custom-file*. If you want to use the Perl API, then select *intermine-items-xml-file*.

Source type	When to use?
db	To load data directly from another **database**
gff	for GFF files
fasta	for FASTA files
obo	for Ontology files
custom-file	If you have a data file and want to parse using **Java**
intermine-items-xml-file	If you have a data file and want to parse using a **language other than Java**
intermine-items-large-xml-file	Same as above but the file is very very large

The script also creates a gradle project if one does not exist.

3. Add your source to your project XML file

Add your new source to the project XML file so it will be run during your build. Below are example project XML snippets for each source type. Note that different parser types have different expected parameters.

See *Project XML* for further reading about the project XML file.

Versions The "version" provided for each source has to match the version of your data parser, e.g. you would want to set *version=1.2.3* for your source *bio-source-mysource-1.2.3.jar*. If you do not provide a version in the project XML file, the default InterMine version will be used.

See `/database/data-sources/custom/dataparser-versions` for details.

3. Write your parser

For most types of data, you'll have to write some code to store your data into InterMine.

Note: Run *make_source* with no arguments to get a full list of source types.

custom-file This a source that reads from a file in a custom format. A custom Java FileConverter will be needed. The *make_source* script will create a skeleton *FileConverter* in *<source-name>/src/main/java/org/intermine/bio/dataconversion*. Edit this code to process the particular file you need to load, using the *Java Items API* to create and store items to the database.

The *project.xml* configuration is as below:

```
<!-- add to your mine's project XML file -->
<source name="my-new-source-name" type="my-new-source-name" version="1.2.3">
  <property name="src.data.dir" location="/some/data/directory"/>
  <!-- optionally specify includes or excludes -->
  <property name="src.data.dir.includes" value="*.xml"/>
</source>
```

See `/database/data-sources/custom/dataparser-versions` for details on how to version your data parser.

Additional Properties in Project XML Any properties you define in a source entry in your mine's project.xml will be available in that source's converter or post-processing class, providing that there is a setter with an appropriate name.

This applies to any class that inherits from:

- org.intermine.dataconversion.DataConverter
- org.intermine.dataconversion.DBConverter
- org.intermine.dataconversion.DirectoryConverter
- org.intermine.dataconversion.FileConverter
- org.intermine.postprocess.PostProcessor

For instance, if you have this entry:

```
<!-- in project XML -->
<source name="my-new-source-name" type="my-new-source-name" version="2.3.4">
  <property name="bar.info" value="baz"/>
  <property name="bazMoreInfo" value="hello-world"/>
</source>
```

Then those values will be available (provided you create the setters correctly):

```
// In a class that extends org.intermine.postprocess.PostProcessor, for example
public void setBarInfo(String info) {
  // given the example project XML values above, "info" has the value of "baz"
  this.info = info;
}
public void setBazMoreInfo(String moreInfo) {
  // given the example project XML values above, "moreInfo" has the value of "hello-world"
  this.moreInfo = moreInfo;
}
```

intermine-items-xml-file This type of source can read a file in InterMine Items XML format and store the data in a mine. The *project.xml* configuration is as below:

```
# add your source to your project XML file
<source name="my-new-source-name" type="my-new-source-name" version="1.2.3">
  <property name="src.data.file" location="/some/directory/objects_in_intermine_format.xml"/>
</source>
```

See this page for more information on the Items XML format and links to APIs that can generate it. This source type doesn't generate any stub Java code.

intermine-items-large-xml-file This source works as above but writes the XML to an intermediate items database to avoid reading the whole file into memory at once. This is the best choice for large XML files where large is several hundred megabytes (although this depends on the amount of RAM specified in your *GRADLE_OPTS* environment variable).

db This source reads directly from a relational database, it will generate a skeleton *DBConverter* in *<source-name>/src/main/java/org/intermine/bio/dataconversion*. You will use the Java API to store data to the InterMine database.

To connect to the original database you need to add properties in xxxmine.properties with the prefix *db.sourcename*.

This is tested for PostgreSQL and MySQL.

Common properties (to be added to your mine properties file):

```
db.sourcename.datasource.dataSourceName=db.sourcename
db.sourcename.datasource.maxConnections=10
db.sourcename.datasource.serverName=SERVER_NAME
db.sourcename.datasource.databaseName=DB_NAME
db.sourcename.datasource.user=USER_NAME
db.sourcename.datasource.password=USER_PASSWORD
```

Add these for PostgreSQL:

```
db.sourcename.datasource.class=com.zaxxer.hikari.HikariDataSource
db.sourcename.datasource.dataSourceClassName=org.postgresql.ds.PGSimpleDataSource
db.sourcename.driver=org.postgresql.Driver
db.sourcename.platform=PostgreSQL
```

Add these for MySQL:

```
db.sourcename.datasource.class=com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
db.sourcename.driver=com.mysql.jdbc.Driver
db.sourcename.platform=MySQL
```

The db value has to match the *source.db.name* in your project XML entry, for example:

```
# project XML
<source name="chado-db-flybase-dmel" type="chado-db" version="1.2.3">
  <property name="source.db.name" value="flybase"/>
  ...
</source>
```

Example entry in flymine.properties:

```
# flymine.properties
db.flybase.datasource.class=com.zaxxer.hikari.HikariDataSource
db.flybase.datasource.dataSourceClassName=org.postgresql.ds.PGSimpleDataSource
db.flybase.datasource.dataSourceName=db.flybase
db.flybase.datasource.serverName=localhost
db.flybase.datasource.databaseName=FB2011_01
db.flybase.datasource.user=USERNAME
db.flybase.datasource.password=SECRET
db.flybase.datasource.maxConnections=10
db.flybase.driver=org.postgresql.Driver
db.flybase.platform=PostgreSQL
```

GFF3 Create a gff source to load genome annotation in GFF3 format. This creates an empty *GFF3RecordHandler* in *<source-name>/src/main/java/org/intermine/bio/dataconversion*. The source will work without any changes but you can edit the *GFF3RecordHandler* to read specific attributes from the last column of the GFF3 file. See the InterMine tutorial and *GFF3* for more information on integrating GFF3.

FASTA Create a fasta source to load sequence data in FASTA format. This creates an empty **FastaConverter.java* file in *<source-name>/src/main/java/org/intermine/bio/dataconversion*. The source will work without any changes but you can edit the **FastaConverter.java* to read specific attributes from the fasta file. See the InterMine tutorial and *FASTA* for more information on integrating FASTA.

OBO Create a obo source to load ontology in OBO format.

```
# an example OBO entry
<source name="go" type="go">
  <property name="src.data.file" location="/data/go/go.obo" version="1.2.3"/>
</source>
```

You don't need to write any code to parse the OBO file, the ontology terms are created automatically.

4. Update the Additions file

Update the file in the *src/main/resources* directory called *new-source_additions.xml*. This file details any extensions needed to the data model to store data from this source, everything else is automatically generated from the model description so this is all we need to do to add to the model. The file is in the same format as a complete Model description.

To add to an existing class the contents should be similar to the example code below. The class name is a class already in the model, the attribute name is the name of the new field to be added and the type describes the type of data to be stored. In the example the *Protein* class will be extended to include a new attribute called *extraData* which will hold data as a string.

```xml
<?xml version="1.0"?>
<classes>
  <class name="Protein>" is-interface="true">
    <attribute name="extraData" type="java.lang.String"/>
  </class>
</classes>
```

To create a new class the *new-source_additions.xml* file should include contents similar to the example below:

```xml
<?xml version="1.0"?>
<classes>
  <class name="NewFeature" extends="SequenceFeature" is-interface="true">
    <attribute name="identifier" type="java.lang.String"/>
    <attribute name="confidence" type="java.lang.Double"/>
  </class>
</classes>
```

The extends clause is optional and is used to inherit (include all the attributes of) an existing class, in this case we extend *SequenceFeature*, an InterMine class that represents any genome feature. *is-interface* should always be set to true. The attribute lines as before define the names and types of data to be stored. A new class will be created with the name *NewFeature* that extends *SequenceFeature*.

To cross reference this with another class, similar XML should be used as the example below:

```xml
<class name="NewFeature" extends="SequenceFeature" is-interface="true">
  <reference name="protein" referenced-type="Protein" reverse-reference="features"/>
</class>
```

In the example above the we create a link from NewFeature to the Protein class via the reference named protein. To complete the link a reverse reference may be added to Protein to point back at the NewFeature, this is optional - the reference could be one-way. Here we define a collection called features, this means that for every NewFeature that references a Protein, that protein will include it in its features collection. Note that as this is a collection a Protein can link to multiple NewFeatures but NewFeature.protein is a reference so each can only link to one Protein.

The reverse entry needs to be added to Protein (still in the same file):

```xml
<class name="Protein" is-interface="true">
  <collection name="features"  referenced-type="NewFeature" reverse-reference="protein"/>
</class>
```

The final additions XML should look like:

```xml
<?xml version="1.0"?>
<classes>
  <class name="Protein>" is-interface="true">
    <attribute name="extraData" type="java.lang.String"/>
    <collection name="features"  referenced-type="NewFeature" reverse-reference="protein"/>
  </class>
  <class name="NewFeature" extends="SequenceFeature" is-interface="true">
    <attribute name="identifier" type="java.lang.String"/>
    <attribute name="confidence" type="java.lang.Double"/>
    <reference name="protein" referenced-type="Protein" reverse-reference="features"/>
  </class>
</classes>
```

If all the data you wish to load is already modelled in InterMine then you don't need an additions file. See *Model Description* for details.

Global Additions File If you don't want to create an additions file for each of your mine's data sources, you can also create a "global" additions file. See the "Global Additions File" section of *Model Merging* for details on how to set this parameter.

5. Update Keys file

Within the *src/main/resources* directory is a file called *new-source_keys.properties*. Here we can define primary keys that will be used to integrate data from this source with any exiting objects in the database. We want to integrate genes by their *primaryIdentifier* attribute so we define that this source should use the key:

```
# new-source_keys.properties
Gene.key_primaryidentifier=primaryIdentifier
```

See *Model Merging*

6. Build your JAR and put on the classpath

Now your code is ready, compile it, build a JAR and put on the classpath with this command:

```
./gradlew install
```

See the "Version" section above for how to properly version your JAR.

Note: This JAR is now on your classpath. If you make changes, you will want to run this command again.

7. Run a build and load your data!

Once you've updated the config files, and written your parser (if necessary), create the database as usual. The source should now be included when building the mine.

```
./gradlew builddb
```

Note: Run the *clean* task before *builddb* when changing the model. *clean* removes the *build* directory which is the location of the data model. If you don't, you won't see your new data model changes!

It's also recommended that you write a unit test for your source. It saves time!

InterMine Items XML

InterMine items XML is a generic format that encodes data the matches InterMine class definitions.

```
<items>
   <item id="0_1" class="NewFeature" implements="">
      <attribute name="identifier" value="feature2"/>
      <attribute name="confidence" value="0.8"/>
      <reference name="protein" ref_id="0_3"/>
  </item>
  <item id="0_2" class="NewFeature" implements="">
```

```
        <attribute name="identifier" value="feature2"/>
        <attribute name="confidence" value="0.37"/>
        <reference name="protein" ref_id="0_3"/>
    </item>
    <item id="0_3" class="Protein" implements="">
        <attribute name="primaryAccession" value="Q8I5D2" />
        <attribute name="extraData" value="proteinInfo"/>
        <collection name="features">
            <reference ref_id="0_1" />
            <reference ref_id="0_2" />
        </collection>
    </item>
</items>
```

Here, the root element is always <items>.

Within <items> each object has is within a separate <item> element.

Each <item> has an id with the format <NAMESPACE_SUBID>. For simple cases, the namespace can always be '0'. These IDs are used to signify connections between items within the item XML file - once the data is loaded into InterMine its own serial IDs are used instead and these Item XML ids disappear.

The child elements of an <item> are either

- <attribute> - this has the name of the attribute (matching the defined class name) and a value
- <reference> - where the property is a reference to some other item by its Items XML id.
- <collection> - this is a collection of <reference>s

Example scripts used to generate InterMine Items XML can be found at intermine_items_example.pl.

Datatypes

The data formats required for attributes in InterMine Items XML for the most part they are fairly obvious and match internal Java types (e.g. strings are UTF-8, doubles are 64-bit IEEE 754 floating point).

One exception is the format required for Dates. InterMine allows this to be expressed in 3 different ways.

1. As the number of seconds since the Unix epoch.
2. In the string format 'yyyy-MM-dd HH:mm:ss', assuming UTC.
3. In the string format 'yyyy-MM-dd', assuming UTC.

If parsing fails for all these formats then InterMine will throw a RuntimeException.

APIs

InterMine Items XML can either be generated directly in your favourite programming language, or there are a number of language-specific APIs that can generate it, and handle issues like Item XML allocation and referencing automatically.

Java Items API 'Items' are a data format for the InterMine system, each Item represents a Java data object that will be stored. They are a convenient way to deal with data that is portable to other languages and has a simple XML format.

This API is currently available only within a *DataConverter* running internally within an InterMine source (i.e. it can't yet be used entirely separately from InterMine)

Usage in a Converter Items are most commonly used in a converter which provides some convenience methods.

Items are usually manipulated in a converter as part of a source InterMine source. All converters subclass *DataConverter* or one of its subclasses. This provides some convenience methods.

Create an item - this uses an *ItemFactory* (see below) which validates the class name and all fields against the data model:

```
Item gene = createItem("Gene");
```

Store an item (or collection of items) to the target items database:

```
store(gene);
store(items);
```

For a simple example of a converter see the *wormbase-identifiers* converter.

Item methods Item has methods to set values of attributes, references to other objects and collections of other objects.

To set an attribute (a field of an Item that is a Java type, e.g. String, Integer) use *setAttribute*). Note that all attribute types are treated as a String, they will be parsed to the appropriate type later.

```
gene.setAttribute("symbol", "zen");
organism.setAttribute("taxonId", "7227");
```

All items have an identifier generated for them automatically, these are used to reference other Items. You can set a reference with to an Item identifier or by using the item itself.

```
String orgIdentifier = organism.getIdentifier();
gene.setReference("organism", orgIdentifier);
```

Or:

```
gene.setReference("organism", organism);
```

Set collections of other Items:

```
List<Item> publications = new ArrayList<Item>();
publications.add(pub1);
publications.add(pub2);
gene.setCollection(publications);
```

Or add one at a time:

```
gene.addToCollection("publications", pub1);
gene.addToCollection("publications", pub2.getIdentifier());
```

Attribute, Reference and ReferenceList (collections) can all be created independently and added to Items, this is sometimes useful in parsers to avoid holding too many Items in memory.

Creating Items with an ItemFactory When not used in a Converter you should create Items using an ItemFactory (the Converter does this for you), this validates the class name and all attribute/reference names against the data model. This requires that you get a Model instance (if there isn't already one).

```
Model model = Model.getInstance("genomic");
ItemFactory factory = new ItemFactory(model);
```

Create an item with the class name.

```
Item gene = itemFactory.makeItemForClass("Gene");
Item organism = itemFactory.makeItemForClass("Organism");
```

Reading/Writing XML To write a collection of Items to XML use *FullRenderer*:

```
FileWriter fw = new FileWriter(new File(fileName));
fw.write(FullRenderer.render(items));
fw.close();
```

To read an XML file into a List of items use *FullParser*:

```
List items = FullRenderer.parse(new FileInputStream(file));
```

Perl Items API In the *intermine/perl* directory we provide a Perl library for creating files in InterMine "Item XML" format. Files in this format can be loaded into an InterMine database by creating a "source".

Usage Most code using these modules will follow this pattern:

Make a model

```
my $model = InterMine::Model->new(file => $model_file);
```

Make a new InterMine item XML document:

```
my $document = InterMine::Item::Document->new(
  model  => $model,
  output => $out_file,
);
```

Make an item:

```
my $gene = $factory->make_item("Gene");
```

Set some attributes

```
$gene->set(identifier => "CG10811");
```

or references:

```
my $org = $factory->make_item("Organism");
$org->set(taxonId => 7227);
$gene->set(organism => $org);
```

or collections:

```
$gene->set(transcripts => [$transcript1, $transcript2]);
```

It is also possible to combine creation and attribute setting in one command:

```
my $gene = $factory->make_item(
  'Gene',
  identifier  => 'CG10811',
  organism    => $org,
  transcripts => [$transcript1, $transcript2],
);
```

Repeat 4 as necessary then call *$document->write* to write the items to the output.

FlyMine example Example using the FlyMine model:

```
use InterMine::Model;
use InterMine::Item::Document;

my $model_file = $ARGV[0] or die;

my $model    = InterMine::Model->new(file => $model_file);
my $document = InterMine::Item::Document->new(model => $model);

my $organism = $document->add_item(
    'Organism',
    taxonId => 7227,
);

my $pub1 = $document->add_item(
    'Publication',
    pubMedId => 11700288,
);
my $pub2 = $document->add_item(
    'Publication',
    pubMedId => 16496002,
);

my $gene = $document->add_item(
    'Gene',
    identifier   => "CG10811",
    organism     => $organism,
    publications => [$pub1, $pub2]
);

# write as InterMine Items XML
$document->write();
```

Output:

```
<items>
   <item id="0_4" class="" implements="Gene">
      <attribute name="identifier" value="CG10811" />
      <collection name="publications">
         <reference ref_id="0_2" />
         <reference ref_id="0_3" />
      </collection>
      <reference name="organism" ref_id="0_1" />
   </item>
   <item id="0_1" class="" implements="Organism">
      <attribute name="taxonId" value="7227" />
   </item>
   <item id="0_2" class="" implements="Publication">
      <attribute name="pubMedId" value="11700288" />
   </item>
   <item id="0_3" class="" implements="Publication">
      <attribute name="pubMedId" value="16496002" />
   </item>
</items>
```

Example In the InterMine *scripts* repository there is a longer example: intermine_items_example.pl

The script has three arguments:

- a string describing a *DataSet*
- a taxon id
- the path to a genomic model file

If you install XML::Writer, the script should run as:

Example command line: .. code-block:: perl

./intermine_items_example.pl "FlyMine" 5833 flymine/dbmodel/resources/main/genomic_model.xml

Python Items API A prototype Python items API is available at https://github.com/synbiomine/synbiomine-et/tree/master/modules/python/intermyne.

Usage examples are at

- https://github.com/synbiomine/synbiomine-et/blob/master/sources/eggnog/eggNog-v4p5-2ItemsXML.py
- https://github.com/synbiomine/synbiomine-et/blob/master/sources/ecocyc/ecocyc_pathways_flat_files_2itemsXML.py
- https://github.com/synbiomine/synbiomine-et/blob/master/sources/parts/synbis/synbisParts2ItemsXML.py

Id Resolvers

The ID resolver uses the files in the specified directory to create a large map. The key for the map is the unique identifier (the MOD ID, for example the MGI:, RGD, FBgn, ZFIN: identifiers). The values in the map are all the symbols, old identifiers, dbxrefs (e.g. Ensembl).

unique gene identifier	symbol, name, ensembl ID ...
MGI:97490	pax6, paired box gene 6 ...

The ID resolver then uses this map to replace old or non-unique identifiers with the unique identifier. This allows genes to be merged correctly into the database, and lets each mine be interoperable with other friendly mines.

The ID resolver is used in several data sources, Homologene for example.

If you look at the Homologene data, you'll see they don't use the MGI identifier. See:

1212	10090	18508	Pax6	7305369 NP_038655.1
1212	10116	25509	Pax6	6981334 NP_037133.1

When parsing the Homologene data file, the ID resolver replaces the symbol "Pax6" with the MGI identifier. The parser sets MGI:97490 to be the primary identifier then stores the gene to the database. Similarly, it replaces Pax6 with "RGD:3258" for the rat gene. And so on.

ID resolvers available in InterMine

EntrezGeneIdResolverFactory	NCBI gene info for a collection of organisms	ftp://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz
FlyBaseIdResolverFactory	flybase chado db, for ''D.melanogaster" only	ftp://ftp.flybase.net/releases/current/psql flybase chado
WormBaseChadoIdResolverFactory	wormbase chado db, for ''C.elegans" only	modENCODE specific
ZfinIdentifiersResolverFactory	zebrafish ids	http://zfin.org/downloads/identifiersForIntermine.txt
MgiIdentifiersResolverFactory	mouse ids	ftp://ftp.informatics.jax.org/pub/reports/MRK_List2.rpt
RgdIdentifiersResolverFactory	rat ids	ftp://rgd.mcw.edu/pub/data_release/GENES_RAT.txt
HgncIdResolverFactory	HGNC human gene ids	http://www.genenames.org/cgi-bin/hgnc_downloads.cgi
EnsemblIdResolverFactory	Ensembl id	customised
HumanIdResolverFactory	human ids	customised

Using ID Resolvers in InterMine data converters

Many data converters use the Entrez (NCBI) Gene ID resolver:

1. Download the identifier file - ftp://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz
2. Unzip the file to */DATA_DIR/ncbi/gene_info*
3. Create a sub directory */DATA_DIR/idresolver/* as file root path and a symbolic link *entrez* to the file

```
$ cd /DATA_DIR/idresolver/
$ ln -s /DATA_DIR/ncbi/gene_info entrez
```

4. Add the root path to the file in *~/.intermine/MINE.properties*

```
resolver.file.rootpath=/DATA_DIR/idresolver/
```

Id resolvers and corresponding symbolic to data file:

Resolver	Symbolic link
EntrezGeneIdResolverFactory	entrez
WormBaseChadoIdResolverFactory	wormid
ZfinIdentifiersResolverFactory	zfin
MgiIdentifiersResolverFactory	mgi
RgdIdentifiersResolverFactory	rgd
HgncIdResolverFactory	hgnc
EnsemblIdResolverFactory	ensembl
HumanIdResolverFactory	humangene

In the data converter, the ID resolver is given an identifier. The resolver then looks in the map for the identifier.

number of matches	returns
0	NULL
1	new identifier
>1	NULL

Using ID Resolvers in your data converters

A factory will find data root path from *~/.intermine/MINE_NAME.properties*, path needs to be absolute.

```
resolver.file.rootpath=/DATA_DIR/idresolver/
```

the key and the symbolic link of the data file need to be hard-coded in factory class, e.g. in *EntrezGeneIdResolverFactory*

```
private final String propKey = "resolver.file.rootpath";
private final String resolverFileSymbo = "entrez";
```

As for database case, e.g. flybase chado

```
# chado DB for flybase data

db.flybase.datasource.class=org.postgresql.jdbc3.Jdbc3PoolingDataSource
db.flybase.datasource.dataSourceName=db.flybase
db.flybase.datasource.serverName=NAME
db.flybase.datasource.databaseName=DBNAME
db.flybase.datasource.user=USER
db.flybase.datasource.password=PWD
db.flybase.datasource.maxConnections=10
db.flybase.driver=org.postgresql.Driver
db.flybase.platform=PostgreSQL
```

the key also needs to be hard-coded in factory class, e.g. in FlyBaseIdResolverFactory

```
private final String propName = "db.flybase";
```

Configuration The Entrez gene identifier source has a configuration file, *entrezIdResolver_config.properties*. You shouldn't have to edit this file.

This config will parse fruit fly identifiers, e.g. FLYBASE:FBgn0088803

```
7227.primaryIdentifier.xref=FLYBASE
```

If you don't want to strip the prefix from the identifier, use this config:

```
10116.primaryIdentifier.prefix=RGD:
10090.primaryIdentifier.prefix=MGI:
```

Warning: The EBI changed how they format their data. If you have a recent data file, you do NOT want the above configuration for MGI.

To replace a taxonomy identifier with a strain, use the following:

```
4932.strains=559292
```

To ignore certain organisms, do this:

```
taxon.ignored = 7165,6239
```

IdResolverService IdResolverService is a java class providing static methods to get id resolver directly. It's also the most straight forward way to create an id resolver. For example, to create a fish id resolver by taxon id in a converter:

```
IdResolver rslvr = IdResolverService.getIdResolverByOrganism("7955");
```

You can use the IdResolverService to create resolver by taxon id, a list of taxon ids, or by organism, e.g.

```
IdResolver flyRslvr = IdResolverService.getFlyIdResolver();
```

Resolve an Id As the resolver maintains java maps of one or more organisms' identifiers, you must explicitly tell it which organism you want it to resolve for, e.g.

```
String pid = flyRslvr.resolveId(taxonId, identifier).iterator().next();
```

It is also possible there are two or more matching primary identifiers for a particular identifier, in this case, discard this identifier, e.g.

```
int resCount = flyRslvr.countResolutions(taxonId, identifier);
if (resCount  = 1) {
  LOG.info("RESOLVER: failed to resolve fly gene to one identifier, ignoring gene: "
          + identifier + " count: " + resCount + " FBgn: "
          + flyRslvr.resolveId(taxonId, identifier));
  return null;
}
```

Writing a New ID resolver

An IdResolver factory will create an IdResolver which will read and parse data from a file or database containing identifier information, to save them to a Java map which will be writen to a cached file.

The new factory class need to inherit super class IdResolverFactory:

```
public class HumanIdResolverFactory extends IdResolverFactory
```

createIdResolver method:

```
// 1. check if the resolver which has the taxon and class has already been created
resolver.hasTaxonAndClassName(taxonId, this.clsCol.iterator().next())

// 2. Restore cached data from file. New data will be append to the cached file.
boolean isCachedIdResolverRestored = restoreFromFile();

// 3. To implement reading and parsing data from a customized file/db, see createFromFile method and
```

createFromFile method:

```
// Ref HumanIdResolverFactory.java
// Parse a tab delimited file. Add to resolver.
String symbol = line[0];

resolver.addMainIds(taxonId, symbol, Collections.singleton(symbol));
```

createFromDb method:

```
// Ref FlyBaseIdResolverFactory.java
// 1. Set db connection parameters in MINE.properties, scroll up to see flybase chado setting.
// 2. Connect to the database and query the data.
// 3. Parse ResultSet, addIdsFromResultSet method
```

Multiple taxon ids:

```
// Ref EntrezGeneIdResolverFactory.java
public IdResolver getIdResolver(Set<String> taxonIds) {
      if (taxonIds == null || taxonIds.isEmpty()) {
          return null;
      }
      return getIdResolver(taxonIds, true);
}
```

Multiple classes:

```
// Ref FlyBaseIdResolverFactory.java
public FlyBaseIdResolverFactory(Set<String> clsCol) {
    // clsCol is set in parent class IdResolverFactory.java
    this.clsCol = clsCol;
}
```

Multiple files or mixture of file and db:

```
// We don't have an example to handle muliple files, but one can always add them and parse them one b
// We have an example of handling db and file together, ref WormBaseIdResolverFactory.java
```

Add resolver factory to IdResolverService:

```
// Ref IdResolverService.java
public static IdResolver getHumanIdResolver() {
    return new HumanIdResolverFactory().getIdResolver(false);
}

public static IdResolver getHumanIdResolver(boolean failOnError) {
    return new HumanIdResolverFactory().getIdResolver(failOnError);
}
```

Future Plans

- generalized resolver factory which will read a configuration file to be aware identifier information by column. e.g. type=tab, column.0=mainId, etc.

Data Licences

You are using InterMine to integrate several data sets into a single database, for ease of querying for your end users. It's important that you make it very clear to your users how the data in your mine is licenced and how it can be re-used.

New DataSet.licence field

In InterMine 4.0, we've added *licence* to the "data set" model as a text field. This column is meant to be a **URL** to point to the standard data licence, e.g. https://creativecommons.org/licenses/by/4.0/

```
<!-- InterMine 4.0.0 data model -->
<class name="DataSet" is-interface="true" term="http://semanticscience.org/resource/SIO_000089">
    <!-- licence is a new text field -->
    <attribute name="licence" type="java.lang.String" term="http://purl.org/dc/terms/license"/>
    ...
</class>
```

How is this information being used?

These data can be displayed prominently on the report page and in query results. We'll also use the licences in the RDF generation.

Why does it have to be a URL to a standard data licence?

The contents of *DataSet.licence* should a URL that points to a standard data licence.

Why can't I put a URL to the fair use policy? If you put a URL to the data source's fair use policy for example, the URL might change. Also, sometimes the fair use policy is vague, contradictory or just hard to understand. It's better to only use standard data licences.

Why can't I put a short snippet about the fair use policy for these data? If you summarise the fair use policy, there is a danger that you get it wrong, or the data policy changes.

Providing no information about the data licence is better than having bad information about the data licence.

How to add licence to an InterMine?

If you want to add a licence to your datasets in your mine, you can do so by updating the associated data source that loads that data set.

Core data sources InterMine core data parsers either parse a standard file type, e.g. FASTA, GFF or a specific file type from a specific data source, e.g. OMIM, UniProt

Standard file types

To update the data licence, add the licence information to the project XML file. An example:

```
<!-- gff example -->
<source name="my-gff" type="my-gff" version="4.0.0">
  <!-- add licence here -->
  <property name="gff3.licence" value="https://creativecommons.org/licenses/by-sa/3.0/" />
  ...
</source>
```

FASTA

```
<!-- FASTA example -->
<source name="my-fasta" type="fasta">
  <!-- add licence here -->
  <property name="fasta.licence" value="https://creativecommons.org/licenses/by/4.0/"/>
  ...
</source>
```

NB: The prefix has to match the *type* of the data source.

OBO

```
<!-- OBO example -->
<source name="so" type="so">
  <property name="src.data.file" location="so.obo"/>
  <!-- add licence here -->
  <property name="licence" value="https://creativecommons.org/licenses/by/4.0/"/>
</source>
```

All others

We've updated all InterMine core data sources with the correct data licence. This requires no action from you. Use the library as normal, and the data parser will populate the *DataSet.licence* field.

However, not every core data source has a data licence. About 1/3 of the data sets InterMine has libraries for have data licences. The rest only have text about fair use. We hope that as data licences become more popular and visible, this number will rise.

Your data sources DataSet now has a licence field, so you will want to update this field in your data parser.

Here is an example using the Java API:

```
// set the licence using the Java API in your data parsers
private static final String LICENCE = "https://creativecommons.org/licenses/by/4.0/";
Item dataSet = createItem("DataSet");
dataSet.setAttribute("licence", licence);
```

If you are using the *BioFileConverter*, you can use the constructor like so:

```
// add data licence
super(writer, model, DATA_SOURCE_NAME, DATASET_TITLE, "http://www.gnu.org/licenses/gpl.txt");
```

This will update the data set licence field for you.

None of my data sources have data licences

We discovered that only a minority of data sets have a licence: of the 26 core data set types that InterMine supports, only 9 have a data set licence, although 14 had some text about fair use.

Please see our blog posts for more details.

1.5.3 Database Building

A 'build' of a mine is a complete data loading run starting from an empty database. It is recommended that you use the *project_build script*. The build script runs the data integration and any post-processing steps.

Each mine has an integrate project that reads the project.xml file and builds the data warehouse. This steps through each *source* defined in the project.xml file and transates the specified data from a source format and loads it into the production database. Data integration is governed by primary keys, any conflicts are resolved by a priorities config file.

project_build script

To run a full build of InterMine, you must use the *project_build* script. This is a Perl program that reads a project.xml file and loads each source in turn. This makes multiple calls to Gradle to avoid memory problems encountered when running many Java task sequentially from Gradle. It also has the option of dumping the production database during the build and recovering from these dumps in case of problems.

Note: This script requires the Expect and XML::Parser::PerlSAX Text::Glob perl modules - install with: *sudo cpan -i XML::Parser::PerlSAX Expect Text::Glob*

Download the file from the intermine-scripts repository:

```
flymine $ wget https://raw.githubusercontent.com/intermine/intermine-scripts/master/project_build
```

Run the build script from the mine directory:

```
flymine $ ./project_build -b -v server_name /some/dump/location/dump_file_prefix
```

The *server_name* is hostname of the machine where the *pg_dump* command should be run. If you are running *project_build* on the same machine as PostgreSQL then you should specify *localhost* as the server name. If the PostgreSQL server is on a remote machine, give its hostname. In that case the script will try to run *pg_dump* on the remote machine using *ssh*. This makes dumping a little faster and allows for the case where */some/dump/location/dump_file_prefix* is only visible on the remote machine.

Dumps are performed when a source has *dump=true* in its *project.xml* definition:

```
<source name="uniprot-malaria" type="uniprot" dump="true">
  <property name="uniprot.organisms" value="36329"/>
  <property name="src.data.dir" location="/data/flyminebuild/malaria/uniprot/7.7/36329"/>
</source>
```

In this example, the dump will be made immediately after the *uniprot-malaria* source has been ''successfully" merged.

Once all sources are integrated *project_build* will run any post-processing steps (also configured in the *project.xml*).

It is also possible to run individual integrate and post-process steps separately, see below.

Command line options

The *project_build* script accepts the following flags:

-v	is passed to ant to make it run in verbose mode, ant output can be seen in *pbuild.log*
-l	attempt to restart by reading the last dump file (see note below)
-b	run build-db before starting build and drop any existing backup databases (created when using the -t flag)
-V	set the release number to pass to gradle (as -Prelease=release_number)

Dump files take the name *dump_file_prefix*.final.

Running project_build with ''*-l*" will reload the latest dump (if any) with *dump_file_prefix* and restart the build from that point.

Note: You must use the full path to the dump file, e.g. */some/dump/location/dump_file_prefix*

Running a Single Datasource

Before starting the build process you will need to set up the appropriate properties and then initialise your database with this command:

```
flymine $ ./gradlew builddb
```

Warning: Running the *builddb* target will drop the current database and create a new, blank database.

To run a data source, run this command in your mine directory, specifying the source name (as it appears in project.xml):

```
flymine $ ./gradlew integrate -Psource=uniprot --stacktrace
```

Most sources have multiple stages in retrieving data, to run just one stage use:

```
flymine $ ./gradlew integrate -Psource=uniprot -Paction=load --stacktrace
```

The stages are:

preretrieve pre-processing that is done

retrieve load data from source database/files into an items database

load read from a target items database and integrate into the production database

See */system-requirements/software/gradle/index* for the full list of common Gradle tasks, or run *./gradlew tasks* to see the list of available tasks on the command line.

Running a Custom Datasource

The build script expects the data source to be on the classpath already. If you are using a data source provided by InterMine, that parser will be put on the classpath for you. If you are using a custom source, you will need to put it on the classpath yourself. You can use the Gradle Maven plugin task *install* to compile your Java code, build the JAR and put on your classpath.

```
# run the install task to build your JAR
flymine-bio-sources $ ./gradlew install

# you can install a single source
flymine-bio-sources $ ./gradlew rnai:install
```

The *install* task will place the JAR in the Maven directory "~/.m2/repository".

Project XML

This document describes the InterMine project XML file. This file is located in the mine directory and determines:

- the Mine's data model
- which data sources are loaded during a build

The project XML file has two sections:

<sources>

The <*source*> elements list and configure the data sources to be loaded, each one has a *type* that corresponds to the name of the bio-source artifact (jar) which includes parsers to retrieve data and information on how it will be integrated. The *name* can be anything and can be the same as *type*, using a more specific name allows you to define specific integration keys. Each source also has a *version*. If one is not provided, the default InterMine version will be used.

<*source*> elements can have several properties: *src.data.dir*, *src.data.file* and *src.data.includes* are all used to define locations of files that the source should load. Different parser types accept different properties, see the two links below for the full list and example project XML entries.

- For details on the project XML specific data sources, see the individual sources page at *Data Source Library*.

- For details on how to write a project XML for a custom source, see *Writing your own data source*

<post-processing>

Specific operations can be performed on the Mine once data is loaded, these are listed here as <*post-process*> elements. For details on which postprocesses are available, see *Post processing*.

Versions

Each data source has its own version. See `/database/data-sources/custom/dataparser-versions` for details on how to version your own data sources and how to specify which versions to use.

Data model

The data model is generated by iterating though each project listed in the project XML file and retrieving its additions.xml file. This file is then merged into the other additions files. There is also an optional "Global" additions file, see *Model Merging* for details.

Examples

For an example project XML file, see Biotestmine's project.xml file.

Data Integration

Data integration works by using keys for each class of object to define equivalence for objects of that class. For example:

- *primaryIdentifier* is used as a key for *Gene*
- *taxonId* is used as a key for *Organism*

For each *Gene* object loaded, a query is performed in the database to find any existing *Gene* objects with the same *primaryIdentifier*. If any are found, fields from both objects are merged and the resulting object stored.

Many performance optimisation steps are applied to this process. We don't actually run a query for each object loaded, requests are batched and queries can be avoided completely if the system can work out no integration will be needed.

We may also load data from some other source that provides information about genes but doesn't use the identifier scheme we have chosen for *primaryIdentifier*. Instead it only knows about the *symbol*, in that case we would want that source to use the *symbol* to define equivalence for *Gene*.

Important points:

- A *primary key* defines a field or fields of a class that can be used to search for equivalent objects
- Multiple primary keys can be defined for a class, sources can use different keys for a class if they provide different identifiers
- One source can use multiple primary keys for a class if the objects of that class don't consistently have the same identifier type
- *null* - if a source has no value for a field that is defined as a primary key then the key is not used and the data is loaded without being integrated.

See *Primary Keys* for more information.

Model Merging

An InterMine model describes the classes available to the InterMine system and their relationships. The model is used to generate the database tables, the Java class files and the web application.

A model can be described using a model file. The model can be either read from one file or built up from several files using "model merging". An example of a single file model is used in the "testmine".

Configuration

An InterMine datamine is built from sources. Each source can contribute to the data model and also provides data. When a mine is built with the *./gradlew builddb* command, the model is created from small "additions" file contributed by each source. Specifically, the model is created by reading the *project.xml* file and merging the model fragment from each addition file for each source.

Other additions files (ie. not from sources) can be explicitly merged by setting the *extra.model.paths.start* and *extra.model.paths.end* properties in the *project.properties* of your *dbmodel* directory. An example from FlyMine's *build.gradle* is:

```
mineDBModelConfig {
  modelName = "genomic"
  extraModelsStart = "so_additions.xml genomic_additions.xml"
  extraModelsEnd = "flybase-chado-db_additions.xml chado-db-stock_additions.xml"
}
```

Here *genomic_additions.xml* and *so_additions.xml* will be merged first and *flybase-chado-db_additions.xml' and 'chado-db-stock_additions.xml* will be merged after all other model fragments.

Note that bio-model's *core.xml* model fragment is always used as a base for the merging - everything will be merge into the classes in *core.xml*

Example

From *core.xml*:

```
...
<class name="Protein" extends="BioEntity" is-interface="true">
  <attribute name="name" type="java.lang.String"/>
  <attribute name="primaryAccession" type="java.lang.String"/>
  <attribute name="length" type="java.lang.Integer"/>
  <attribute name="molecularWeight" type="java.lang.Integer"/>
  <reference name="sequence" referenced-type="Sequence"/>
  <collection name="genes" referenced-type="Gene" ordered="true" reverse-reference="proteins"/>
</class>
...
```

From the uniprot source (*uniprot_additions.xml*):

```
...
<class name="Protein" is-interface="true">
  <attribute name="description" type="java.lang.String"/>
  <attribute name="ecNumber" type="java.lang.String"/>
  <collection name="publications" referenced-type="Publication"/>
</class>
...
```

Final, merged, model definition:

```
...
<class name="Protein" extends="BioEntity" is-interface="true">
  <attribute name="description" type="java.lang.String"/>
  <attribute name="ecNumber" type="java.lang.String"/>
  <attribute name="name" type="java.lang.String"/>
  <attribute name="primaryAccession" type="java.lang.String"/>
  <attribute name="length" type="java.lang.Integer"/>
  <attribute name="molecularWeight" type="java.lang.Integer"/>
  <reference name="sequence" referenced-type="Sequence"/>
  <collection name="publications" referenced-type="Publication"/>
  <collection name="genes" referenced-type="Gene" ordered="true" reverse-reference="proteins"/>
</class>
...
```

The resulting class has all attributes of the *Protein* from *core.xml* and from *uniprot_additions.xml*. Note that in uniprot we don't need to declare a base class for *Protein* (like as *extends="BioEntity"*) as the base class from *core.xml* is merged into the final class.

Global Additions File

Previously the data model was merged from all data sources' additions XML file (plus the SO terms, core and genomic additons). This is no longer true. Since each source is in its own JAR now, the data model is self-contained for each data source. Therefore if you reference a class in your data parser, it must be present in the additions file.

Alternatively, you can specify a single data model file that will be merged into each source:

```
// Place this in build.gradle in root of your mine-bio-sources directory, e.g. flymine-bio-sources/bu
//
// Must be in the subprojects {} section of the build.gradle file
//
// bioSourceDBModelConfig {
//    # file should be in the root of your mine-bio-sources directory
//    globalAdditionsFile = "MY-MINE_additions.xml"
// }
```

This setting will merge the specified additions file, e.g. *MY-MINE_additions.xml*, into the data model for everyone of your mine's data sources.

Primary Keys

This document describes the configuration used by the InterMine integration system to identify objects that are identical to each other. Two objects are deemed to be identical if they have matching fields for at least one primary key used for the class of object. Primary keys are defined in the resources directory of the data source, and should be called "$DATA-SOURCE-NAME_keys.properties".

Data source keys configuration files

For each data source, there is a properties file providing a list of the primary keys that can be used when integrating that data source. The file lists the primary keys by name for each class. When loading objects of a particular class the keys define which fields should be used to look up in the database for existing objects to merge with.

The keys are specified in each source in a file: *$DATA-SOURCE-NAME/src/main/resources/$DATA-SOURCE-NAME_keys.properties*.

These *_keys.properties* files define keys in the format:

```
Class.name_of_key = field1, field2
```

The *name_of_key* can be any string but you must use different names if defining more than one key for the same class, for example in *uniprot_keys.properties* there are two different keys defined for *Gene*:

```
Gene.key_primaryidentifier = primaryIdentifier
Gene.key_secondaryidentifier = secondaryIdentifier
```

Use common names for identical keys between sources as this will help avoid duplicating database indexes. Postgres uses the key names to create indexes.

Each key should list one or more fields that can be a combination of *attributes* of the class specified or *references* to other classes - in which case there should be a key defined for the referenced class as well.

Warning: The build system will use any valid key it finds - so be careful! e.g. if you have keys for BioEntity and SequenceFeature and Gene in your keys file, any of the three keys may be used to merge a Gene into the database.

It is still possible to use a legacy method of configuring keys, where keys are defined centrally in *db-model/resources/genomic_keyDefs.properties* and referenced in source *$DATA-SOURCE-NAME_keys.properties* files.

Global primary key configuration file [DEPRECATED]

Warning: This is an older method of defining keys in a central configuration file. Use the method described above instead.

Define keys in a central file [DEPRECATED] This file is a Java properties file, so all the data is in form of single lines of the form “property name = property value”. A line is a comment if it begins with a hash character, and blank lines may be present. This file defines a set of primary keys by name for each class. Defining a primary key on a class makes it apply to all the subclasses too. This file should be located in *MINE_NAME/dbmodel/resources*.

To define a primary key, enter a line in the following form:

```
# <name of model>_keyDefs.properties file in MINE_NAME/dbmodel/resources
Classname.primary_key_name = field1, field2
```

This line means that the class “Classname” and all its subclasses have a primary key called “primary_key_name” that matches two objects if the values of both of the fields “field1” and “field2” are identical. Only attributes and references can be used as fields in a primary key, not collections.

Here is a short example of the configuration file. The configuration file we use for the FlyMine system is a good example.

```
# some keys defined in flymine/dbmodel/resources/genomic_keyDefs.properties
Gene.key_identifier_org=identifier, organism
Gene.key_symbol_org=symbol, organism
Gene.key_organismdbid=organismDbId
Gene.key_ncbiGeneNumber=ncbiGeneNumber
Protein.key_identifier_org=identifier, organism
Protein.key_primaryacc=primaryAccession
```

Using keys (from central file) in each source [DEPRECATED] The properties file for each data source lists primary key names from the the central *genomic_keyDefs.properties* file. The file lists the primary keys by name for each class; the primary key names must be defined in the global keyDefs file mentioned in the previous section. If a class is not mentioned, then instances of that class will never be merged with other objects. For each class, there should be a line like the following:

```
# keys file in SOURCE/resources that references keys defined in global keyDefs properties file.
Gene = key_identifier_org, key_symbol_org
```

This line means that the class “Gene” and all its subclasses have a two primary keys available for this data source, called “key_identifier_org” and “key_symbol_org”, which should be defined properly in the global configuration.

Warning: This is an older method of defining keys in a central configuration file. Use the method described in the first section instead.

Priority Configuration

This document describes the format of the configuration file used by the InterMine system’s integration to resolve conflicts between different data originating from different data sources. This file should be created as *MINE_NAME/dbmodel/resources/MODEL_NAME_priorities.properties*

When two objects from two different data sources have been identified as equivalent by the PrimaryKeys, those two objects must then be merged into a single object. It is possible that the different data sources may give different values for some of the fields of the objects, so the integration system must choose between the two values. This could be implemented as a manual data curation step, but we decided to make it automatic, by allowing data sources to be placed in priority order on a per-field basis. This means that if two data sources have a conflicting value for a field, the data source with the highest priority for that field will supply the value used in the final object.

If you think that a particular field will never have conflicting values supplied by different data sources, then it need not be mentioned in the priority configuration. However, if there is a priority configured, it must list all the data sources that will provide values for that field. A value of null is ignored as “not a value”. A wildcard of “*” matches all data sources that aren’t otherwise listed, which can be useful to reduce the size of the priorities file and the number of times it needs to be updated.

File format

The file must be called “MODEL_NAME_priorities.properties” and be in the classpath of the data loader. The configuration file is a Java properties file, so lines beginning with a hash character are comments, and blank lines are allowed. To specify a priority for all the fields of a particular class (and its subclasses), create a line in the file like this:

```
Classname = datasource1, datasource2
```

However, individual fields can be separately specified too. The order of lines in the file does not matter. Create a line like this:

```
Classname.fieldname = datasource2, datasource1
```

Alternatively, you can use a wildcard, for instance to say that datasource1 is right all the time:

```
Classname = datasource1, *
```

Or to say that all the datasources provide the correct value, except datasource1:

```
Classname = *, datasource1
```

The data sources are listed in order of decreasing priority. Note that all the field values controlled by a wildcard must be identical, so for instance datasource2 and datasource3 must not conflict in any of these examples. Some example files are our testing priorities file and our FlyMine priorities file.

Class Hierarchy

Because this is an object-oriented database, classes of object are arranged in a class hierarchy, that is some classes are sub-classes of other super-classes. Therefore, it is possible to define a priority on Gene.name and on BioEntity.name, which refer to the same attribute. The priority system will only work if the priorities are completely unambiguous. That is, Gene.name and BioEntity.name must be set to the same thing, or an error will be reported. Generally, you should only configure one of those two classes.

Validation

The configuration will be validated at the beginning of data loading. The validation will check that no configuration is made for classes that do not exist, and for data sources which do not exist (which could easily be a typo), and that no class hierarchy problems exist. Note that there is an extremely small chance that some class hierarchy problems may be spotted after validation with some extremely exotic data, but we do not expect to ever see such data.

Post processing

Some operations are performed on the integrated data before the webapp is released - post-processing. For example, setting sequences of SequenceFeatures, filling in additional references and collections or transferring orthologues from translations to genes. These are steps that run after the data loading is completed. They are used to set calculate/set fields that are difficult to do when data loading or that require multiple sources to be loaded. Some postprocessing steps are core to InterMine.bio and should always be run, others are contributed by particular sources.

Post-processing steps are specified in the project XML file and run from the mine:

```
~/git/flymine $ ./gradlew postprocess --stacktrace
```

To run individual post-process steps use, for example:

```
~/git/flymine $ ./gradlew postprocess -Pprocess=do-sources --stacktrace
```

When running one postprocess step like this (multiple steps separated by comma is not supported), the *-Pprocess* used must match an *post-process* in the *post-processing* section of the *project.xml* file.

Post-processing is run automatically after integrating if using the *project_build* script.

To add a post-process step to InterMine, you need to add the Java definition to the project and call the post-process from the *PostProcessOperationsTask* class.

Note: Be sure to put the postprocesses in the correct order. Each task is executed in the order listed on your project XML so be sure to put the webapp tasks last in the last, for example. Take a look at the FlyMine project XML file if you need help.

Sequence Features

create-chromosome-locations-and-lengths For genome features this will set the *chromosome*, *chromosomeLocation* and *length* fields which are added to make querying more convenient. Some parts of the webapp specific to genome features expect *chromosomeLocation* to be filled in.

Should I use it? Yes, if you have loaded genome annotation.

transfer-sequences Where a Chromosome has a sequence this will find genome features located on it that don't have sequence set this will calculate and set the sequence for those features.

Should I use it? Yes, if you have loaded genome annotation without sequence set for all features.

create-references Create shortcut references/collections to make querying more obvious.

Should I use it? Yes, for the moment if you are using standard InterMine sources.

create-utr-references Create shortcut references/collections to make querying more obvious. Read the UTRs collection of MRNA then set the fivePrimeUTR and threePrimeUTR fields with the corresponding UTRs.

Should I use it? Yes, if you think it sounds useful.

create-intron-features If you have loaded genome annotation that include exons but does not specify introns this will create Intron objects and name them appropriately.

Should I use it? If genome annotation you have loaded does not include introns.

make-spanning-locations Create a Location that spans the locations of some child objects. Creates a location for Transcript that is as big as all the exons in its exons collection and a location for gene that's as big as all the transcripts in its transcripts collection.

Should I use it? Only if you don't have locations for Genes or Transcripts loaded from another source.

Overlapping and Flanking Features

create-intergenic-region-features Looks at gene locations on chromosomes and calculates new IntergenicRegion features to represent the intergenic regions. These are useful in combination with overlaps for working out, e.g. binding sites that overlap the upstream intergenic region of a gene. Each Gene gets a reference to its upstream and downstream intergenic regions.

Should I use it? Yes, if you have loaded genome annotation and think IntergenicRegions sound useful.

create-location-overlap-index Create a GIST index on the location table to help with overlap queries.

Should I use it? Yes, if you have genome annotation and would like to query overlaps. You must have bioseg installed unless you are using Postgres 9.2 or later. See *Querying over genomic ranges* for details.

create-bioseg-location-index Deprecated.

Should I use it? No. Use *create-location-overlap-index* instead.

create-overlap-view Replace the *sequencefeatureoverlappingfeatures* table with a view that uses a fast index to calculate the overlaps.

Should I use it? Yes, if you have genome annotation and would like to query overlaps. You must have bioseg installed unless you are using Postgres 9.2 or later. See *Querying over genomic ranges* for details.

create-gene-flanking-features Create features to represent flanking regions of configurable distance either side of gene features. These will be used in overlap queries.

Should I use it? Yes, if you have genome annotation and would like to query flanking regions.

Data

do-sources This searches through all sources included in project.xml and runs post-processing steps if any exist. Looks for the property *postprocessor.class* in the *project.properties* of each source, the class specified should be a subclass of *org.intermine.postprocess.PostProcessor*.

Should I use it? - Yes, if you are using standard InterMine sources, they may have post-processing steps.

Webapp

create-attribute-indexes Create indexes on all attributes to help speed up queries.

Should I use it? Always. It should be run after all post-processing steps that write new records to the database as this step creates indexes for all columns in each table.

create-search-index Creates the lucene search index used by the webapp.

Should I use it? Yes, if you are releasing a webapp.

populate-child-features Populate the SequenceFeature.childFeatures() collection.

Should I use it? Yes, only if you use JBrowse and you want your JBrowse web-service endpoints available (see also *JBrowse* and *Web Services*).

summarise-objectstore Counts of the number of objects of each class and for class fields that have a small number of value, a list of those values. See `/database/database-building/post-processing/objectstore-summary-properties` for more information.

Should I use it? - Always. Run after *create-attribute-indexes* to speed this step up.

create-autocomplete-index Creates the indexes for the fields set to be autocompleted in the ObjectStoreSummaryProperties file.

Should I use it? Yes, if you have a webapp.

Post build updating with SQL triggers

Warning: Please note this is an experimental facility and is subject to a number of caveats (see below). Please always take a backup of your database before trying.

We very much welcome feedback, discussion and additional patches for this. Many thanks to Joe Carlson of the DOE Joint Genome Institute for the idea and implementation!

Requirements

1. InterMine release > 1.7.3
2. plpgsql must be installed in your postgres (select * from pg_language where lanname='plpgsql';) Check the postgreSQL manuals for instructions on installing languages if needed.
3. Backup the database prior to making changes, especially if there are changes that affect foreign keys.

Procedure

Traditionally, once the data for a mine has been built, it can only be updated by a complete rebuild. However, sometimes, after a long loading process, you see that something is not right: perhaps a minor issue such as a typo in a name, or perhaps something more major such as errors in an entire dataset. Rather than rebuilding the entire mine from scratch, a process that can take many hours or even many days, you'd like to make changes to your existing data build.

Making such updates requires co-ordinated changes to a number of InterMine tables. For instance, to update a value, one needs to at least:

1. Update the value in InterMine's table for that object (e.g. the length column in the Gene table).
2. Update the value in InterMine's tables for all the ancestor classes of that object (e.g. the length column in the SequenceFeature table).
3. Update the serialized object in the object column of the intermineobject table.

One way to do this is by installing triggers into the PostgreSQL database that will co-ordinate these updates. InterMine can now generate such triggers if you invoke the ant generate-update-triggers in your mine's dbmodel/ directory like so:

```
cd $MINE
./gradlew generateUpdateTriggers
```

This will generate two SQL files in the dbmodel/build/resources/main/ subdirectory

```
add-update-triggers.sql
remove-update-triggers.sql
```

add-update-triggers.sql contains the SQL triggers necessary to co-ordinate table updates. *remove-update-triggers.sql* contains the removal code. All the triggers have a prefix of *im_*.

Adding triggers To add the triggers just execute add-update-triggers.sql using psql like so

```
psql -f add-update-triggers.sql MINE-NAME
```

You can now do basic create/update/delete operations such as:

- UPDATE organism set genus='Homo" where genus='Homer';
- DELETE FROM organism where commonname='yeti';

The triggers propogate the operations to the superclasses and InterMineObjec tables

Tables have default values supplied for id and class, so you can create new records

- INSERT INTO organism (genus,species) values ('Hello','world');

The id is supplied from a sequence im_post_build_insert_serial which is initially set to the maximum id of InterMineObject.

Once you've completed update operations, you must remove the triggers. Failure to do so may cause interference with InterMine's run time serial use, though this point needs to be clarified.

Removing triggers You can remove triggers by executing the *remove-update-triggers.sql* SQL:

```
psql -f remove-update-triggers.sql MINE-NAME
```

What can't be done (yet) Please note that there are a number of database changes that the triggers CANNOT handle as of yet:

1. Foreign key constraints are not enforced. If you delete a gene, there may still entries in the genesproteins table or a reference to this from the geneid field in the mrna table. Foreign keys are enforced at the application layer. This means whoever is doing the update needs to keep things straight. (This is possible to implement. It may be done in the future.)
2. The tracker table is not updated. If you do an integration step after manual operations and the integrator is trying to update a column value that you inserted manually, the integration step will fail.
3. The clob table cannot be manipulated. Again, this may also be changed in the future.
4. If the id field in InterMineObject has exceeded 2^31 and gone negative, the sequence im_post_build_insert_serial cannot be used in INSERT operations without (probably) colliding with another object. The value of the serial must be set manually in this case.

Debugging

Below are common errors you may encounter when building your InterMine. Please contact us if you continue to have problems!

Given a ProxyReference, but id not in ID Map

Error message

```
java.lang.RuntimeException: Exception while dataloading - to allow multiple errors, set the property
```

Problem This error means the code tried to store an object that was referenced by another object but could not find it. This means you've set a reference to an object, but not stored that referenced object in the database.

Here's an example:

```
// set reference to organism object
gene.setReference("organism", organism);
// store gene object
store(gene)
// never store organism object!
```

Gene now refers to an object that does not exist in the database. To fix, make sure you are storing all the correct objects in your code.

Solution Make sure your code is correct and refers only to objects that are going to be stored in the database.

- Unit tests for this data source should be updated to check for these cases.
- If you are loading XML created by another script, be sure to validate the data before loading.

To find out which object is not being stored, use the *item identifier* listed in the error message – in this case, *6_1083*. Look in the *items* database to determine the values for this object.

```
select * from item where identifier = '6_1083'
```

This gives s the class and item id:

```
implementations | classname | identifier |   id
----------------+-----------+------------+---------
                | Strain    | 6_1083     | 1380031
```

We see this object is a Strain. We now know which type of data is not being stored. We can then look in the attribute table to get the identifier. Using the *id* we can query the attribute table.

```
select * from attribute where itemid = 1380031;
```

This gives us any attributes stored for this object, in our example this gives us the primary identifier.

```
 intermine_value |        name        | itemid
-----------------+--------------------+---------
LS2329           | primaryIdentifier | 1380031
```

1.5.4 Data Integrity Checks

Template Comparison

We have written a script that runs queries against the templates publicly available in a mine or a pair of mines. The purpose of these scripts is to:

- Test that all templates run.
- In the case of multiple mines, check that updates haven't radically changed the results.

The script presents their results on standard out, with the option to have them emailed upon completion of the comparison. To have results emailed out, you should have set up and installed *sendmail* on the machine running the comparison.

The script is located here: https://github.com/intermine/intermine-scripts/blob/master/compare_templates_for_releases.py

Python Script

Dependencies This script will run on *cPython* 2.5-2.7, *pypy* and *jython*. It requires the installation of the intermine client module, which can be installed from http://pypi.python.org PyPi with the following command:

```
$ sudo easy_install intermine
```

Invocation The script can be invoked most simply against a single mine as follows:

```
$ python compare_templates_for_releases.py www.flymine.org/flymine
```

To have results emailed, add your email address:

```
$ python compare_templates_for_releases.py www.flymine.org/flymine you@your.host.org
```

Optionally set a ''from" address:

```
$ python compare_templates_for_releases.py www.flymine.org/flymine you@your.host.org noreply@blackhol
```

Comparing against two mines is as above, except you simply need to add a second service location:

```
python compare_templates_for_releases.py www.flymine.org/flymine beta.flymine.org/beta you@your.host
```

Results

The resulting email will look like this:

```
----------------------------------------------------------------------
-----------------------------------In Both: Diff >= 10%
BDGP_Gene                                          release-beta:     260, release-28.0:      62, diff
ChromLocation_CRMOverlappingTFBindingsite          release-beta:      42, release-28.0:     213, diff

----------------------------------------------------------------------
-----------------------------------Only in 28.0:
ChromosomeLocation_Tiffin                                        8
Disease_GeneOrthologue                                         363
ESTclone_LocationOverlappingGeneOrthologue_new                  93
ESTclone_LocationOverlappingGeneStructure                        4
Gene_Inparalogue                                                11
Gene_Tiffin                                                    156
Probe_Gene                                                       1
TiffinBSmotif_expressionTerm                                    49
TiffinBSmotif_genes                                           1356
TiffinBSmotif_locations                                         23
----------------------------------------------------------------------
-----------------------------------Only in beta:
Amplicon_RNAiResults                                            39
Gene_AdjacentGene_FlyAtlas_downstream                            0
Gene_OverlapppingGenes                                           1
Genes_Publications                                          126002
Organism_interologs                                            278
--------------------------------------------------In Both: Diff < 10%
All_Genes_In_Organism_To_Publications              release-beta:  126002, release-28.0:  121503, diff
AlleleClass_Allele                                 release-beta:    2132, release-28.0:    2117, diff
```

1. ‘‘‘In Both: Diff >= 10%’‘‘ - templates run in both mines and result counts returned were very different.
2. ‘‘‘Only in’‘‘ - template was found in one mine and not the other.
3. ‘‘‘In Both: Diff < 10%’‘‘ - template run in both mines and results returned were different. It’s probably safe to assume these are okay.

Acceptance Tests

How to run the tests

1. Add a file to *MINE_NAME/dbmodel/resources*, eg. *flymine_acceptance_test.conf*
2. run acceptance tests here:

```
~/git/flymine $ ./gradlew runAcceptanceTests
```

The results will be in *MINE_NAME/dbmodel/build/acceptance_test.html*

Types of tests

You can assert that a query returns true:

```
assert {
        sql: select count(*) >= 400000 from goannotation
}
```

Or doesn’t have any results:

```
no-results {
        sql: select * from datasource where url is null or name is null or description is null
        note: all fields of data source should be filled in
}
```

Or has at least some results:

```
some-results {
        sql: select * from organism where name = 'Anopheles gambiae'
        note: We should have an Anopheles gambiae  object but not an Anopheles gambiae PEST one
}
```

1.5.5 InterMine performance

Data loading performance

The speed at which InterMine is able to load data into the databases depends on a number of factors including complexity of objects loaded, hardware specifications and so on. Below are some steps you can take to speed up your build.

Java options

Loading data can be memory intensive so there are some Java options that should be tuned to improve performance. See a note about *Java*

PostgreSQL

- Use a recent, correctly configured version of PostgreSQL.
- InterMine can actually build a database for production faster than Postgres can undump from a backup file. This is because we generate precomputed tables and indexes in parallel using several CPUs simultaneously. Therefore, it makes sense to complete the last few steps of the build (namely precomputed tables and indexes) on your production servers directly, instead of completing them on the build server and transferring the data across to the production servers.

Recommended settings for PostgreSQL are in *Installing PostgreSQL*

Hardware

See a note about *Hardware*

Storing Items in order

When loading objects into the production ObjectStore the order of loading can have a big impact on performance. It is important to store objects before any other objects that reference them. For example, if we have a Gene with a Publication in its evidence collection and a Synonym referencing the Gene, the objects should be stored in the order: Publication, Gene, Synonym. (If e.g. the Gene is stored after the Synonym a placeholder object is stored in the Gene's place which is later replaced by the real Gene. This takes time).

Objects are loaded in the order that Items are stored by converter code or the order they appear in an Items XML file. When Items are stored into the items database (during the build or using *ant -Dsource=sourcename -Daction=retrieve*) you can check if there are improvements possible with this SQL query:

```
SELECT classnamea, name, classnameb, count(*)
FROM (SELECT distinct itema.classname AS classnamea, name, itemb.classname AS classnameb, itemb.ident
      FROM item AS itemA, reference, item AS itemB
      WHERE itema.id = itemid AND refid = itemb.identifier
            AND itema.id < itemb.id) AS a
GROUP BY classnamea, name, classnameb;
```

If there are no results then no improvement can be made. The example below shows that there were 27836 Gene Items stored after the Synonyms that reference them. *subject* is the name of the reference in Synonym. Changing the store order would improve performance.

```
classnamea |  name   | classnameb | count
------------+---------+------------+-------
Synonym    | subject | Gene       | 27836
```

Switching off the DataTracker

In order to allow data conflicts to be managed, the system needs to keep track of where each piece of data came from. This is because conflicting values will be resolved by a priority system where one data source is regarded as more reliable than another for a particular field value. However, storing this data takes significant time while running the DataLoader, and can now be switched off on a per-class basis for the whole DataLoading run. This is useful if you know that there will never be any data conflicts for a particular class and the data will not be merged, e.g. Sequence or Location objects. The configuration is found in the properties file for the project, and a configuration line for “datatrackerMissingClasses” is added to the IntegrationWriter entry, like this:

```
integration.production.class=org.intermine.dataloader.IntegrationWriterDataTrackingImpl
integration.production.osw=osw.production
integration.production.datatrackerMaxSize=100000
integration.production.datatrackerCommitSize=10000
integration.production.datatrackerMissingClasses=OneAttribute
```

The parameter is a comma-separated list of class names for which no tracking data should be stored. All objects which are instances of these classes will be omitted, including subclasses.

Non-InterMineObjects

For the ultimate in performance gain, objects can be stored in the database which are not instances of InterMineObject. Such objects are stored in “flat mode” in an SQL table. Because they do not have an ID, they cannot be referenced by other objects, fetched by ID, or deleted by ID, and they cannot have a collection, or be in a many-to-many collection. They are not stored in the main InterMineObject table, and are not stored in the DataTracker, and are never merged with other objects by the DataLoader. No class hierarchy may exist in these classes, and no dynamic objects may make use of these classes. The objects take much less space in the database than instances of InterMineObject. The objects can however contain attributes and references to other objects, and can be in one-to-many collections of other objects. The full Query interface will work correctly with these simple objects. Simple objects are configured in the Model by declaring the superclass of a class to be “java.lang.Object” in the model description, like this:

```
<class name="SimpleObject" is-interface="false" extends="java.lang.Object">
    <attribute name="name" type="java.lang.String"/>
    <reference name="employee" referenced-type="Employee" reverse-reference="simpleObjects"/>
</class>
```

We recommend you set *is-interface* to "false" for these objects. There is no need to specify these classes in the "dataTrackerMissingClasses" property as above, because these classes are never tracked.

Proxies

In object/relational mapping systems when an object is read from the database we need to know which objects it is related to in order to follow references and collections. However, if the entire object were fetched each time and then it's referenced objects were fetched, etc one request could materialise millions of objects. e.g. if Gene references Organism and has a collection of Proteins we would fetch a Gene, it's Organism and Proteins then recusively fetch all references for the new objects.

Instead we use proxies. *org.intermine.objectstore.proxy.ProxyReference* appears to be a standard *InterMineObject* but in fact just contains an object id, when any method is called on the proxy the object is materialized automatically. e.g. Calling *gene.getOrganism()* returns a *ProxyReference* but calling *gene.getOrganism().getName()* de-refernece s the proxy and returns the name.

org.intermine.objectstore.proxy.ProxyCollection does the same for collections but wraps an objectstore query required to populate the collection, the collection is materialised in batches as it is iterated over by wrapping a SingletonResults object.

Results

Here are the results of trying some of the above so you can see how effective the various strategies are:

.	Load time	objs / min	DB size	tracker size
Original	4.51 min	1,525,015	9.6 GB	3.7 GB
No tracker	3.94 min	1,748,446	5.56 GB	1 GB
Consequence as SimpleObject	3.37 min	2,044,448	4.6 GB	1.4 GB
Both of above	3.20 min	2,153,291	4.1 GB	1 GB

Performance test

In objectstore/test run 'ant test-performance' (requires unittest database, currently on beta branch)

Build server with SATA drives:

```
test-performance:
[run-performance-test] Starting performance test...
[Finalizer] INFO com.zaxxer.hikari.pool.HikariPool - HikariCP pool db.unittest is being shutdown.
[run-performance-test] Stored 10000 employee objects, took: 19870ms
[run-performance-test] Stored 10000 employee objects, took: 15231ms
[run-performance-test] Stored 10000 employee objects, took: 15811ms
[run-performance-test] Total store time: 50912ms. Average time per thousand: 1697.067ms.
[run-performance-test]
[run-performance-test] Reading all employee objects with empty object cache
[run-performance-test] Read  10000 employee objects, took: 720ms.
[run-performance-test] Read  20000 employee objects, took: 272ms.
[run-performance-test] Read  30000 employee objects, took: 230ms.
[run-performance-test] totalTime: 1244 rowCount: 30000
[run-performance-test] Finished reading 30000 employee objects, took: 1244ms. Average time per thous
```

Workstation with SSDs:

```
        [run-performance-test] Starting performance test...
[run-performance-test] Stored 10000 employee objects, took: 8303ms
[run-performance-test] Stored 10000 employee objects, took: 7334ms
[run-performance-test] Stored 10000 employee objects, took: 7727ms
[run-performance-test] Total store time: 23364ms. Average time per thousand: 778.800ms.
[run-performance-test]
[run-performance-test] Reading all employee objects with empty object cache
[run-performance-test] Read  10000 employee objects, took: 444ms.
[run-performance-test] Read  20000 employee objects, took: 126ms.
[run-performance-test] Read  30000 employee objects, took: 101ms.
[run-performance-test] totalTime: 681 rowCount: 30000
[run-performance-test] Finished reading 30000 employee objects, took: 681ms. Average time per thousar
```

You should expect similar.

Query performance (precomputed tables)

InterMine can make use of precomputed tables (analagous to materialised views) for faster execution of queries. These can represent any SQL query (or InterMine query) and can automatically be substituted into incoming queries by our own cost-based query optimiser. For example, a precompute that joins three tables could be used in a larger query that includes that join thus reducing the total number of tables joined in the query. Template queries can be precomputed completely so that for any any value entered in an editable constraint the query will be executed from a single database table.

Template queries

Webapp As the superuser, when you create a new template or edit an existing one there is a 'precompute' link on the MyMine saved templates list. Clicking this will create a precomputed table for just this query. It can take some time to create the tables and requests aren't put in a queue so it is not a good idea to click many of these links at once. The 'precompute' link will change to 'precomputed' once there is a precomputed table created.

Command line Precomputing template queries makes sure that public templates will always run quickly. You can precompute all templates saved as the superuser in your userprofile database from the command line. This checks each template first to see if it is already precomputed.

```
~/git/flymine $ ./gradlew precomputeTemplates
```

Manual specification of queries

You can specify any IQL query to precompute in the file *MINE_NAME/dbmodel/resources/genomic_precompute.properties*. These allow you to design queries you think are likely to be created commonly or be parts of larger queries. It is the place to put queries that will be used in list upload and widgets to ensure they run fast.

```
~/git/flymine $ ./gradlew precomputeQueries
```

Here is an example query:

```
   precompute.query.6 =
SELECT a1_.id AS a3_, a2_.name AS a4_
FROM org.intermine.model.bio.Protein AS a1_, org.intermine.model.bio.Organism AS a2_
WHERE a1_.organism CONTAINS a2_
```

You can also specify the classes involved:

```
precompute.constructquery.20 = Protein organism Organism
```

Dropping precomputed tables

To drop all precomputed tables in a database:

```
~/git/flymine $ ./gradlew dropPrecomputedTables
```

Size of precomputed tables

You can see the names and sizes of all precomputed tables by running this SQL query in psql:

```
SELECT relname,category,pg_size_pretty(pg_relation_size(oid))
FROM pg_class, precompute_index
WHERE relname NOT LIKE 'pg_%' and relname = name
ORDER BY pg_relation_size(oid) DESC;
```

Note that this only lists the table sizes, there may be many indexes associated with each table which may also be large. To see the size of all tables and indexes in the database use:

```
SELECT relname,pg_size_pretty(pg_relation_size(oid))
FROM pg_class
WHERE relname NOT LIKE 'pg_%'
ORDER BY pg_relation_size(oid) DESC;
```

Template Summaries

After the templates are precomputed, they are "summarised". This means any dropdowns for the templates will be updated to only include valid values for that specific templates. How it's done:

- All editable constraints are dropped, non-editable constraints are kept
- Valid values (summaries) for dropdowns are recalculated

For example, if you have a template with an option to select a chromosome, all chromosomes in the database will be displayed. However if you have a non-editable constraint setting the value of the organism to be human, only the human chromosomes will be displayed after summarisation.

FAQs

How do you know what to put in the precomputes file? This is what we did for FlyMine:

1. Common joins to be done, e.g. Gene to protein
2. Widgets - see what queries the widgets are running, add those queries
3. Problem areas being reported, certain queries being slower than expected, e.g. interaction queries

These three things, along with precomputing templates, seems to work best.

Ideally we would have some sort of query profiling and would be able to tell where precomputing helps.

How do you tell if what you put in there is actually helping? When the query is logged, it gives the execution time as well:

> bag tables: 0 ms, generate: 1 ms, optimise: 0 ms, ms, estimate: 9 ms, execute: 61 ms, convert results: 7 ms, extra queries: 0 ms, total: 78 ms, rows: 806

This lets you compare query speeds. You can tell the query used a precomputed table by checking the logs for the prefix *precomp_*

Were all these queries (in the flymine file) created by hand? No. We ran all of our analysis tools on the list analysis page, e.g GO enrichment widget and captured the queries being run via the logs.

PostgreSQL is not using my precomputed table when running a query. Help!

1. You must restart Tomcat after you have created all of the precomputed tables or else your new tables won't be used
2. PostgreSQL uses EXPLAIN to decide which query is fastest. If using your table isn't going to be faster, it won't use it. PostgreSQL may be wrong, but that's how it decides which table to use. See http://www.postgresql.org/docs/9.2/static/using-explain.html for details.

A Log Entry

The LOG records three queries:

1. the IQL (InterMine Query Language) query
2. the generated SQL query
3. the optimised query <– this is where you will see your precomputed tables used

IQL

```
2013-10-30 16:59:24 INFO                                   sqllogger     - (VERBOSE) iql: SELECT DISTINCT
```

Generated SQL

```
generated sql: SELECT DISTINCT a7_.id AS a7_id, a2_.id AS a2_id, a3_.id AS a3_id, a8_.id AS a8_id, a5
```

Optimised sql

```
optimised sql: SELECT DISTINCT P98.a1_id AS a7_id, P98.a3_id AS a2_id, P96.id AS a3_id, a8_.id AS a8_
```

bag tables: 0 ms, generate: 1 ms, optimise: 0 ms, ms, estimate: 14 ms, execute: 11 ms, convert results: 0 ms, extra queries: 27 ms, total: 53 ms, rows: 1

Note the *FROM* clause now includes *precomp_45503*. You can query for this name in the database:

```
select * from precompute_index where name ='precomp_45503';
```

You can also run IQL queries directly in the console:

```
~/git/flymine $ ./gradlew runIQLQuery -Pquery='some IQL'
```

Useful ObjectStore properties

You can configure some parameters to update how queries are handled by setting these in your mine.properties file. If you do not, the default values will be used.

os.query.max-query-parse-time

InterMine includes a cost-based query optimiser that attempts to rewrite SQL queries to make use of precomputed tables. This involved parsing SQL strings into a Java representation, which is normally very fast but if multiple OR constraints are found in large queries can be slow.

There is a timeout to prevent query parsing from taking too long, if the time is exceeded a query will run as normal without possible optimisation. The default can be overridden by setting `os.query.max-query-parse-time` in `*mine.properties` to an integer value defining a number of milliseconds.

Used in *QueryOptimiserContext.java*.

os.query.max-time

When the query is executed, via *ObjectStoreInterMineImpl.executeWithConnection()*, SQL EXPLAIN is run on the generated query. If the estimated time to complete the query is more than the *max-time* parameter set, the query will fail.

Defaults to 100000000 milliseconds.

os.query.max-limit

When the query is executed, via *ObjectStoreInterMineImpl.executeWithConnection()*, SQL EXPLAIN is run on the generated query. If the estimated number of rows is more than the *max-limit* parameter set, the query will fail.

Note this relies on Postgres's statistics being up to date and correct, be sure to run *ANALYSE*.

Defaults to 100000000 rows.

os.query.max-offset

Sets the maximum number of rows available to export.

If the offset for a query is greater than the *os.query.max-offset*, the query will fail to run. See *TableExportAction.checkTable()* for the exact ExportException used.

```
// exception thrown in TableExportAction.checkTable()
if (pt.getExactSize() > pt.getMaxRetrievableIndex()) {
    throw new ExportException("Result is too big for export. "
        + "Table for export can have at the most "
        + pt.getMaxRetrievableIndex() + " rows.");
}
```

Defaults to 100000000 rows.

os.queue-len

<obsolete>

1.6 Guide to Customising your Web Application

1.6.1 Guide to Customising BlueGenes

Content

Certain features of the BlueGenes app are controlled by parameters set in the *web.properties* file. These properties are also used in the current webapp user interface.

purpose	parameters	example
default examples for the ID resolver	bag.example.identifiers	`bag.example.identifiers.protein=Q8T3M3,FBpp0081318,FTZ_DROME` and `bag.example.identifiers=CG9151, FBgn0000099` (one per type)
default separators	list.upload.delimiters	`\n\t,`
default regionsearch	genomicRegionSearch.*	`H. sapiens` (note: please do not use long format, e.g. `Homo sapiens`)
default query builder query	services.defaults.query	`"{ \"from\": \"Gene\", \"select\": [\"secondaryIdentifier\", \"symbol\", \"primaryIdentifier\", \"organism.name\"], \"orderBy\": [{ \"path\": \"secondaryIdentifier\", \"direction\": \"ASC\" }], \"where\": [{ \"path\": \"organism.name\", \"op\": \"=\", \"value\": \"Drosophila melanogaster\", \"code\": \"A\" }] }"`
default keyword search	quickSearch.identifiers	`e.g. PPARG, Insulin, rs876498`

Please see *Features* for details on these parameters.

Environment

BlueGenes uses the following parameters defined in the *~/.intermine/$MINE.properties* file.

purpose	parameters	example
location of JavaScript tools	bluegenes.toolLocation	*/intermine/tools/node_modules/*
base URL for requests to the InterMine instance	webapp.baseurl	*http://www.flymine.org*
path appended to the base URL	webapp.path	*flymine*
name of your InterMine instance as it will be displayed in BlueGenes	project.title	*BioTestMine*

Please see *Database and Web application* for details on this property file.

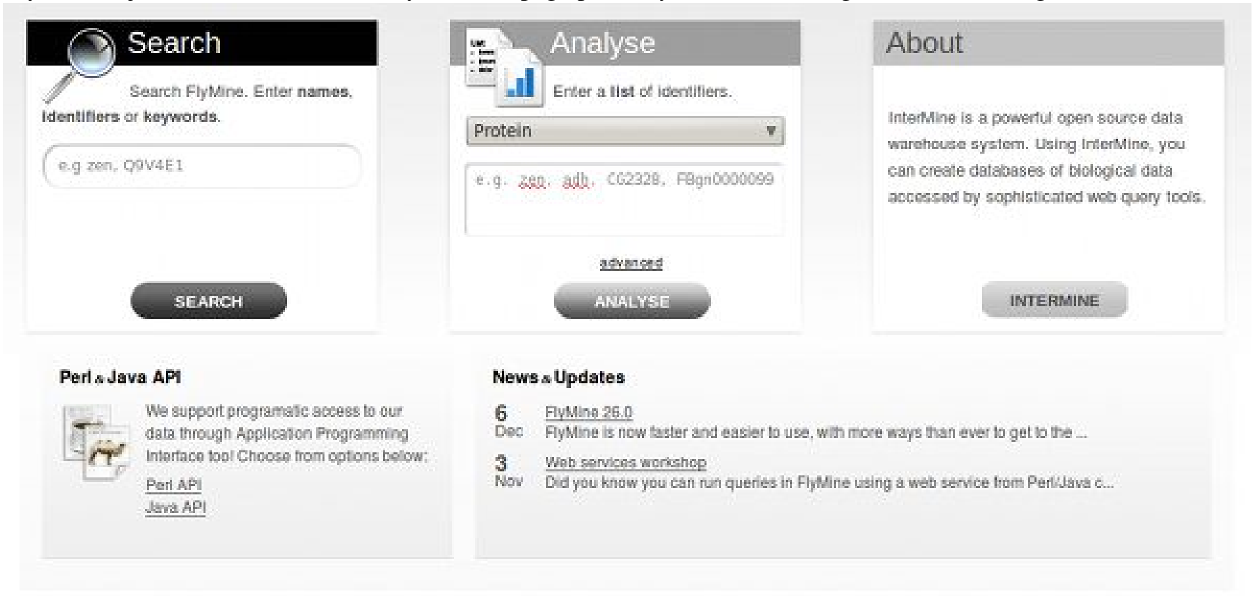
1.6.2 Home page

Note: This text describes how to customize the homepage of your mine.

See also:

General Layout for whole app look & feel.

If you have just installed a new mine, your homepage probably looks something like the following:


Search

Search FlyMine. Enter **names**, **identifiers** or **keywords**.

e.g zen, Q9V4E1

SEARCH

Analyse

Enter a **list** of identifiers.

Protein

e.g. zen, adh, CG2328, FBgn0000099

advanced

ANALYSE

About

InterMine is a powerful open source data warehouse system. Using InterMine, you can create databases of biological data accessed by sophisticated web query tools.

INTERMINE

Perl & Java API

We support programatic access to our data through Application Programming Interface too! Choose from options below:

Perl API

Java API

News & Updates

6 Dec FlyMine 26.0

FlyMine is now faster and easier to use, with more ways than ever to get to the ...

3 Nov Web services workshop

Did you know you can run queries in FlyMine using a web service from Perl/Java c...

In order to do any sort of customizations, one has to add/edit a configuration file for the mine. You will find this file in `webapp/src/main/webapp/WEB-INF/web.properties`.

Open this file in your editor of choice and follow the text below.

Boxes Customization

The three prominent boxes on the homepage will contain a search functionality a list upload functionality and an info box. You can customise the text these contain and the box title.

Search box

The first search box is configured thusly:

```
begin.searchBox.title = Search
begin.searchBox.description = Search FlyMine. Enter <strong>names</strong>, <strong>identifiers</str
or <strong>keywords</strong> for genes, proteins, pathways, ontology terms, authors, etc. (e.g. \
<em>eve</em>, HIPPO_DROME, glycolysis, <em>hb</em> allele).
```

Note: You will find that only the description field accepts HTML.

Second box

```
begin.listBox.title = List Upload
begin.listBox.description = Enter a <strong>list</strong> of identifiers.

bag.example.identifiers=CG9151, FBgn0000099, CG3629, TfIIB, Mad, CG1775, CG2262, TWIST_DROME, \
tinman, runt, E2f, CG8817, FBgn0010433, CG9786, CG1034, ftz, FBgn0024250, FBgn0001251, tll, \
CG1374, CG33473, ato, so, CG16738, tramtrack,  CG2328, gt
```

Third box

The third/info box can contain a descriptive text about your mine or it can offer a link to a tour of the project. Take the example from FlyMine project:

```
begin.thirdBox.title = First Time Here?
begin.thirdBox.description = FlyMine integrates many types of data for <em>Drosophila</em>, \
<em>Anopheles</em> and other organisms. You can run flexible queries, export results and analyse list
data.
begin.thirdBox.link = http://www.flymine.org/help/tour/start.html
begin.thirdBox.linkTitle = Take a tour
```

By providing the .link parameter a button will be shown at the bottom of the box with a custom link of choice.

You can serve up a custom text in the third "information" box to the user, based on whether they have visited the homepage before or not. We do this through a cookie that will, for a year, indicate for your computer, that the homepage has been visited.

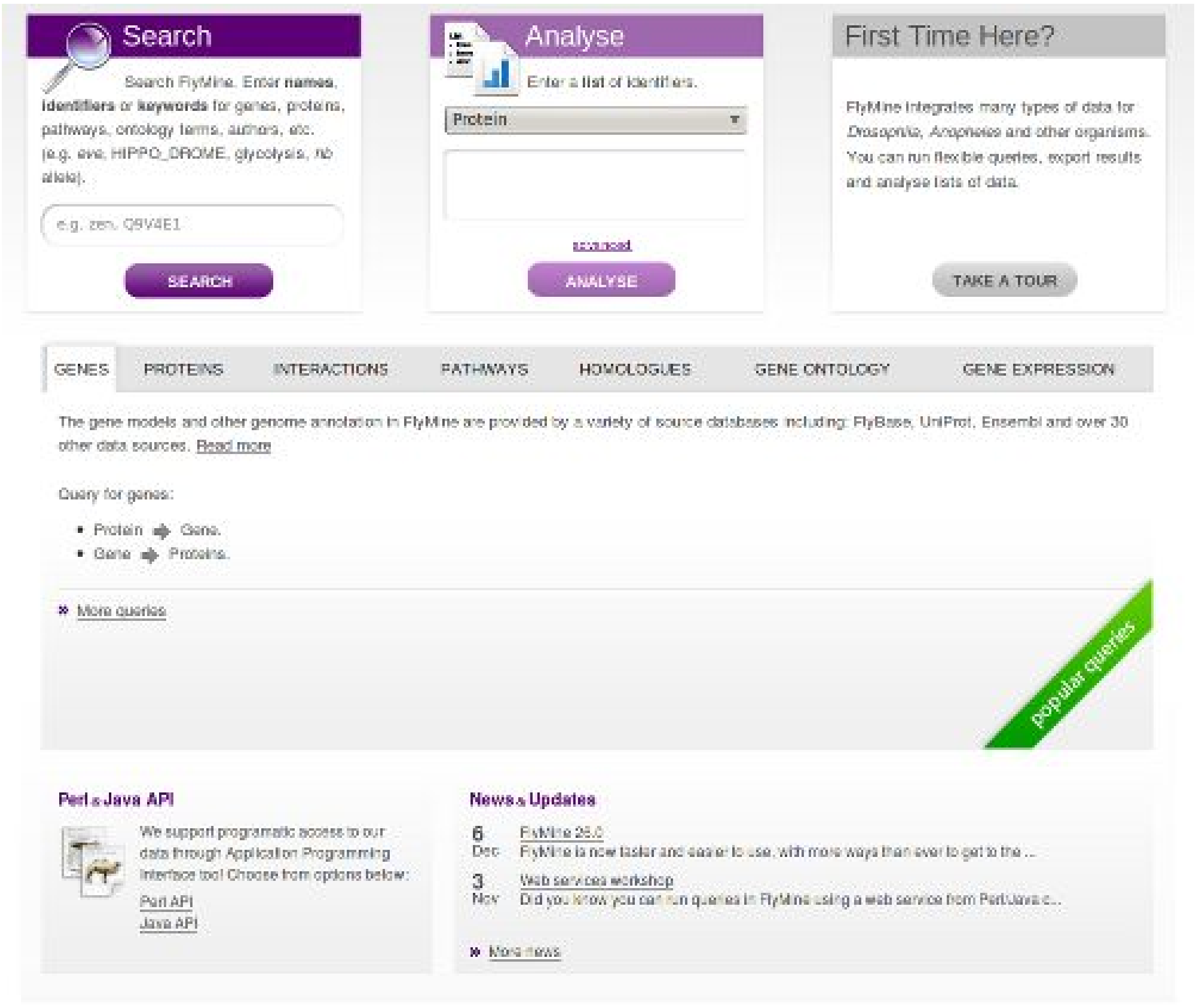
In order to change the values of the third box based on whether the user has visited the page or not, prepend the text "visited" before an uppercased key. For example, if one wanted to say "Welcome Back" instead of "First Time Here?" as the title of the box, we would add the following key=value pair:

```
begin.thirdBox.visitedTitle = Welcome Back
```

The fields that you do NOT set in this way, will simply show the text configured in the normal way. So even though someone has visited the homepage before, unless I add a "visited" property, the text stays the same as before.

Popular Templates Customization

To show the ten most popular template queries per category on your homepage:



Search FlyMine. Enter **names**, **identifiers** or **keywords** for genes, proteins, pathways, ontology terms, authors, etc. (e.g. eve, HIPPO_DROME, glycolysis, *hb* allele).

Enter a **list** of identifiers.

FlyMine integrates many types of data for *Drosophila*, *Anopheles* and other organisms. You can run flexible queries, export results and analyse lists of data.

The gene models and other genome annotation in FlyMine are provided by a variety of source databases including: FlyBase, UniProt, Ensembl and over 30 other data sources. Read more

Query for genes:

- Protein ➔ Gene.
- Gene ➔ Proteins.

We support programatic access to our data through Application Programming Interface too! Choose from options below:
Perl API
Java API

6 Dec FlyMine 26.0
FlyMine is now faster and easier to use, with more ways than ever to get to the ...

3 Nov Web services workshop
Did you know you can run queries in FlyMine using a web service from Perl/Java c...

Example:

```
# web.properties
begin.tabs.1.id = Genomics
```

What this configuration does is it creates a tab on the homepage with (up to) 10 most popular templates from a *Genomics* category. For a template to appear in this section, tag it with the Genomics aspect: *im:aspect:Genomics*.

Note: The tag you apply to the template (e.g. *im:aspect:Genomics*) must match the value of the *id* attribute (e.g. *begin.tabs.1.id = Genomics*).

The number in the config key specifies the order in which we want to show them. So if we have two categories, Genomics and Proteins, and they should appear in this order, we would write this:

```
begin.tabs.1.id = Genomics
begin.tabs.2.id = Proteins
```

The other customisation we can do is specify an informative text that is to appear in the tab above the templates listing (again, this text accepts HTML.):

```
begin.tabs.1.id = Genomics
begin.tabs.1.description = This is some descriptive text
```

The last thing we will show is how to specify a custom category name to show as a link on the tab (entirely optional):

```
begin.tabs.1.id = Genomics
begin.tabs.1.description = This is some descriptive text
begin.tabs.1.name = Genes
```

Example configuration file: FlyMine

Featured Lists

Lists with tag `im:homepage` will be shown on the homepage below the templates listing in a natural order, and/or an order specified by `im:order:n`.

To change the description text associated with this set of lists, edit the properties file like so:

```
begin.listsBox.description = These are the best lists ever
```

RSS/Blog Feed

To add the RSS feed at the bottom right corner of the page, add the following to your MINE properties file (in `.intermine` file):

```
project.rss = http://<your_blog>/<your_feed_url>
```

eg:

```
project.rss=http://blog.flymine.org/?feed=rss2
```

Two latest entries will be shown in the box. If you want to provide a link underneath the entry listing to your blog, add the following to the config file:

```
links.blog = http://<your_blog>
```

1.6.3 Report page

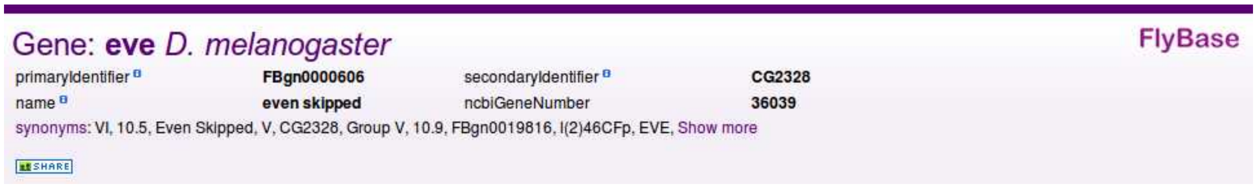
Report Page

Object Title(s)

One can edit the appearance of object title(s) through the `webconfig-model.xml` file (See *Text and messages*).

Let us suppose we want to have a default way of displaying bio entities like gene, protein or probe set. Thus we would look up the entry for bio entity class and add the following configuration:

```
<class className="org.intermine.model.bio.BioEntity">
    <headerconfig>
        <titles>
            <title mainTitles="symbol|primaryIdentifier" numberOfMainTitlesToShow="1" subTitles="*org
        </titles>
    </headerconfig>
</class>
```


Gene: eve D. melanogaster
FlyBase
primaryIdentifier FBgn0000606 secondaryIdentifier CG2328
name even skipped ncbiGeneNumber 36039
synonyms: VI, 10.5, Even Skipped, V, CG2328, Group V, 10.9, FBgn0019816, l(2)46CFp, EVE, Show more

We see that the titles are defined within the headerconfig block. Then we have the following fields:

mainTitles a vertical bar (|) separated list of keys for which we would like to see values.

numberOfMainTitlesToShow (optional) this property is useful if we want to only show a maximum of one value in the title. As per our example the system will first try to resolve the "symbol" of the BioEntity, if it is known, we will show just that. However, if a symbol is not provided, then we try to resolve the primaryIdentifier. The system thus follows left-to-right rule when deciding what and how many fields to show. Main titles will always be bold.

subTitles this is where we define sub titles. Again we can use the vertical bar to define a number of key values to display. Subtitles can be displayed in three ways based on the tags around them that define element formatting:

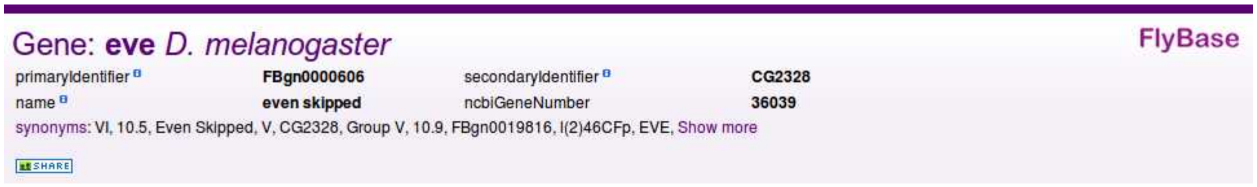
- `primaryIdentifier` (default): the element will be displayed without any formatting applied
- `*primaryIdentifier*`: the element will be in italics, useful for organism names
- `[primaryIdentifier]`: the value will appear in square brackets

Note: Classes of objects *inherit* from their parents, thus unless we provide a different configuration for a, say, Protein title, the formatting from BioEntity will be applied. Fields resolved in the title(s) will be removed from the summary below it.

Custom Header Link

One can have a custom link in the header of the page through the `webconfig-model.xml` file.

```
<headerconfig>
    <customlinks>
        <customlink
            url="http://flybase.org/reports/{primaryIdentifier}.html"
            image="flybase_logo_link.png"
        />
    </customlinks>
</headerconfig>
```


Gene: eve D. melanogaster
FlyBase
primaryIdentifier FBgn0000606 secondaryIdentifier CG2328
name even skipped ncbiGeneNumber 36039
synonyms: VI, 10.5, Even Skipped, V, CG2328, Group V, 10.9, FBgn0019816, l(2)46CFp, EVE, Show more

The example above has been inserted as a child of the Gene class `<class className="org.intermine.model.bio.Gene">`. The parameters are as follows:

url this is where we specify the target of the link. The item in the curly brackets is a variable parameter that will get resolved as a property for the current object.

image (optional) defines a name of the image from "model/images" (e.g.: `webapp/src/main/webapp/model/images`) to resolve.

`text` (optional) defines a link text that will appear (next to an image if provided). The link will then appear in the top right corner of the header. If no image or text is provided, the link text will default to the URL.

Note: The order the fields appear in your webconfig-model is the order in which they will appear on the report page (left to right).

References & Collections

Each object has a number of fields configured in the model, like `length` or `proteins` for Gene. The first is a reference to a single value or an object, the latter is a list of values/objects. These then appear on the report page as References and Collections.

To configure in which category on the page these are to show, follow *Website Admin*.

Additionally, one can decide to either show the old style "inline tables" or use the new Results Tables JS library when displaying these. To enable the latter, edit your *web.properties* as follows:

```
inline.collections.in.tables = true
```

This will display any inline collections in table widgets. Inline collections appear expanded by default and can be manually collapsed by the user. To make all inline collections appear as collapsed, add or edit the following property in your *web.properties*:

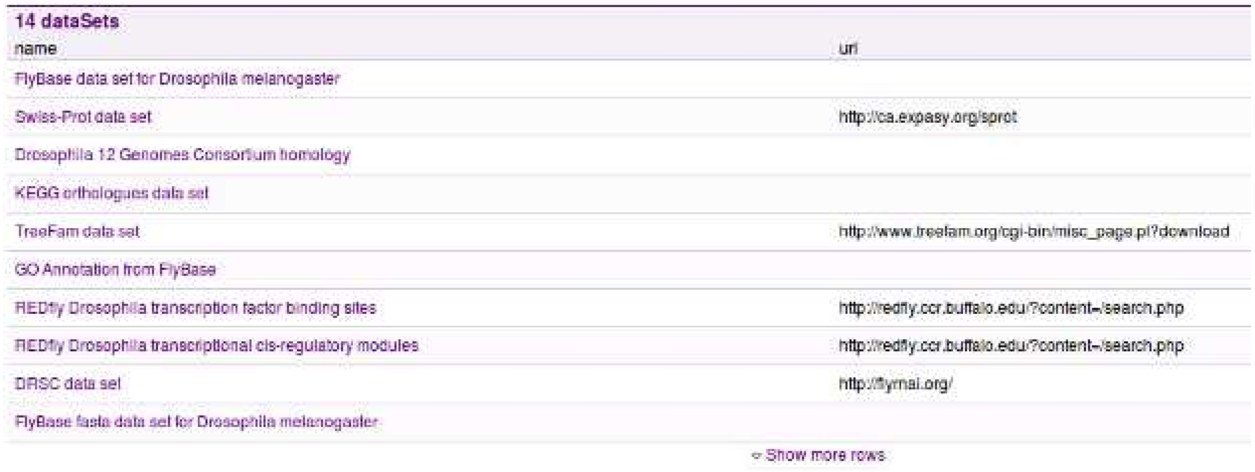
```
web.collections.expandonload=false
```

If `use.localstorage` is `true`, and localStorage is available, then a particular collection's expanded or collapsed state will be remembered and not overriden by the default state property.

```
use.localstorage = true
```

Inline Lists

Inline lists are lists of values in one column of a report page table, displayed in one line. Take ''dataSets" on a Gene object as follows:



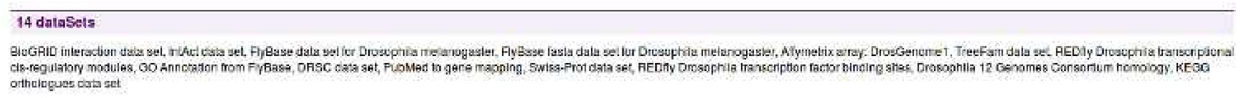
14 dataSets

name	url
FlyBase data set for Drosophila melanogaster	
Swiss-Prot data set	http://ca.expasy.org/sprot
Drosophila 12 Genomes Consortium homology	
KEGG orthologues data set	
TreeFam data set	http://www.treefam.org/cgi-bin/misc_page.pl?download
GO Annotation from FlyBase	
REDfly Drosophila transcription factor binding sites	http://redfly.ccr.buffalo.edu/?content=/search.php
REDfly Drosophila transcriptional cis-regulatory modules	http://redfly.ccr.buffalo.edu/?content=/search.php
DRSC data set	http://flyrnai.org/
FlyBase fasta data set for Drosophila melanogaster	

Perhaps we would like to only display the names of data sets available. Then, we would add the following to the Gene class (`<class className="org.intermine.model.bio.Gene">`) in the `webconfig-model.xml` file:

```
<inlinelist>
    <table path="dataSets.name" />
</inlinelist>
```

The result:



14 dataSets

BioGRID interaction data set, IntAct data set, FlyBase data set for Drosophila melanogaster, FlyBase fasta data set for Drosophila melanogaster, Affymetrix array: DrosGenome1, TreeFam data set, REDfly Drosophila transcriptional cis-regulatory modules, GO Annotation from FlyBase, DRSC data set, PubMed to gene mapping, Swiss-Prot data set, REDfly Drosophila transcription factor binding sites, Drosophila 12 Genomes Consortium homology, KEGG orthologues data set

Let's go through the available configuration:

`path` refers to the reference or collection and a key (separated by a dot) that refers to a column (key) we want to display.

`showInHeader` (optional) a boolean attribute that, if set to true, will make the list appear in the header section of the page.

`lineLength` (optional) defines the number of characters we would like to show in the list, any content after this length will be revealed upon clicking an "Expand" link. Bear in mind that the line length will not be exact as it will not split words in half.

`showLinksToObjects` (optional) by specifying this boolean attribute we can create links from the list to the individual objects the list refers to.

If we have not set an inline list to appear in the header section of the page, the list will, by default appear in the category "Other" on the report page. If we login to the system we can then (through the report page tagging interface that is revealed to us) tag the list to appear in a specific section.

Custom Displayers

See *Report Displayers* for details on how to create displayers for the report page.

Templates

Tag template with the `im:report` tag. See *Website Admin*.

The template needs to have only one where clause involving the class of the object. You also need to specify an aspect whithin the report page where the template will appear (e.g. `im:aspect:Genomics`)

Templates appear collapsed by default. To make all templates appear expanded when a report page is loaded, add or edit the following property in your *web.properties*:

```
web.templates.expandonload=true
```

As with collections (see above), if `use.localstorage` is enabled and available, then a particular template's expanded or collapsed state will be remembered and not overriden by the default state property.

Warning: The underlying query that populates a template is executed as the template is expanded. Setting `web.templates.expandonload` to `true` can cause a significant increase in a report page's load time, particularly if there are more than a handful of templates.

External Links

See the External Link section of *Features*

Data

See *Data and Widget Configuration* for details on how to change the names of class and fields.

You can also hide collections by tagging them with the `im:hidden` tag.

Report Displayers

See also:

Report Displayers Examples.

Report displayers allow custom display of particular data types on report pages (only), typically to replace default tables with more appropriate presentation of data. Widgets:

1. Use a simple framework to add a JSP for display and optionally Java code to run queries, hold caches, etc.
2. Are assigned to the **summary** section at the top of the page or a particular **data category**
3. Can **replace fields** from the report page to override default display of attributes or collections
4. Are configured in the `webconfig-model.xml` file in your Mine

This page describes how to configure your Mine to include widgets for common data types and how to create your own custom widget.

Configuring displayers

Configuration is placed in a `<reportdisplayers>` section of `webconfig-model.xml`:

```
<reportdisplayers>
    <reportdisplayer javaClass="org.intermine.bio.web.displayer.GeneOntologyDisplayer"
        jspName="model/geneOntologyDisplayer.jsp"
        replacesFields="goAnnotation,ontologyAnnotations"
        placement="Function"
        types="Gene"/>
</reportdisplayers>
```

`javaClass` an optional Java class to run before display, typically this performs database queries or creates data structures used by the JSP. The class should extend `ReportDisplayer` and implement a `display()` method.

`jspName` the JSP file used to display output

`replacesFields` a comma separated list of fields that should not appear on the report page when the displayer is used

`showImmediately` set to `true` to display the displayer immediately as the page loads, without waiting (`false` by default)

`placement` the section on the report page the displayer should appear in, can be 'summary' or a valid data category name.

`types` a comma separated list of class names for this displayer can be used

`parameters` this is a JSON string used to pass arbitrary parameters to particular displayers, you can make use of this for detailed configuration of any displayers you write. For example, the HomologueDisplayer.java is passed a list of data sets to displayer homologues from: `parameters="{'dataSets': ['TreeFam data set', 'KEGG orthologues data set']}"`.

Useful displayers

There are several displayers for common data types that may be useful in many Mines. To enable these just copy the configuration from FlyMine's webconfig-model.xml.

For examples of the common displayers and configuration details please see *Report Displayers Examples*..

Creating a new Displayer

If you've loaded some new data into your Mine or have some great ideas about presenting data from the common data loaders you can create a new displayer. Here are brief instructions, take a look at the many examples for more details.

1. Create a Java class [5] in your mine, e.g. /displayers that inherits from `org.intermine.web.displayer.ReportDisplayer`.
2. Implement `public void display(HttpServletRequest request, ReportObject reportObject)` to perform any queries or processing required and put results on the request.
3. Create a JSP file in *webapp/src/main/webapp/model* to display the results.
4. Add configuration to *webapp/src/main/webapp/WEB-INF/webconfig-model.xml* to set up the `javaClass` and `jspName` created above and set the `types` for which the displayer should appear and the *summary* or a data category (aspect) as the `placement` for the displayer. Optionally set any fields in the report page that should be hidden when this displayer is used.

Troubleshooting As we use AJAX to load the displayers to speed up the initial load of a Report page, JavaScript calls to when a document is ready are executed immediately as the page has finished loading already. Specifically when using GoogleCharts API, one needs to amend the initial loading code with a callback like for example so:

```
google.load("visualization", "1", {"packages": ["corechart"], "callback": drawFlyAtlasChart});
```

Report Displayers Examples

Report displayers you can use in your own Mine and some examples created for specific data types in modMine, FlyMine and metabolicMine.

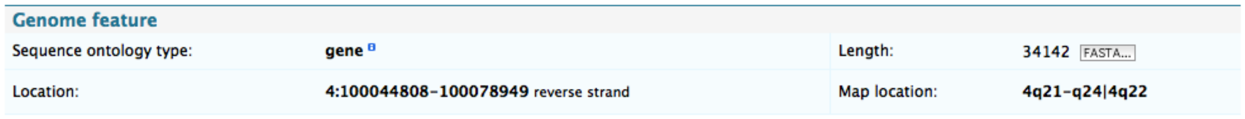
The following displayers can all be used for data loaded by standard InterMine parsers. To see how to configure them check out FlyMine's `webconfig-model.xml`.

SequenceFeature summary

Applicable for any SequenceFeature - shows length, sequence export, chromosome location, cyto location and SO term (where present).

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.SequenceFeatureDisplayer"
    jspName="model/sequenceFeatureDisplayer.jsp"
    replacesFields="chromosome,chromosomeLocation,sequence,length,sequenceOntologyTerm,locations,cyto
    placement="summary"
    types="SequenceFeature"/>
```

[5] ReportDisplayer makes available a variable called `im` which is the `InterMineAPI` which provides access to config and query execution classes.



Genome feature

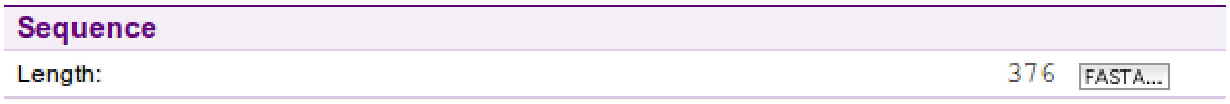
Sequence ontology type:	gene	Length:	34142 FASTA...
Location:	4:100044808-100078949 reverse strand	Map location:	4q21-q24|4q22

Figure 1.8: A Sequence feature displayer in metabolicMine.

Protein sequence

Applicable for Protein - shows length, sequence export.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.ProteinSequenceDisplayer"
    jspName="model/proteinSequenceDisplayer.jsp"
    replacesFields="sequence"
    placement="summary"
    types="Protein"/>
```



Sequence

Length:	376 FASTA...

Figure 1.9: A Protein sequence displayer in FlyMine.

GBrowse

Show an inline image from a configured GBrowse instance.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.GBrowseDisplayer"
    jspName="model/gbrowseDisplayer.jsp"
    replacesFields=""
    placement="Genes"
    types="SequenceFeature"/>
```

This also needs two properties to be configured in the `minename.properties` file: `gbrowse.prefix` and `gbrowse_image.prefix` which give the location of a running GBrowse instance.

```
gbrowse.prefix=http://www.flymine.org/cgi-bin/gbrowse
gbrowse_image.prefix=http://www.flymine.org/cgi-bin/gbrowse_img
```

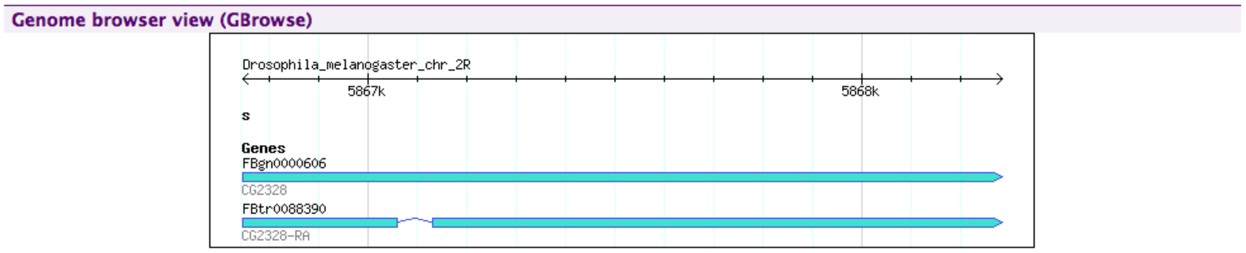
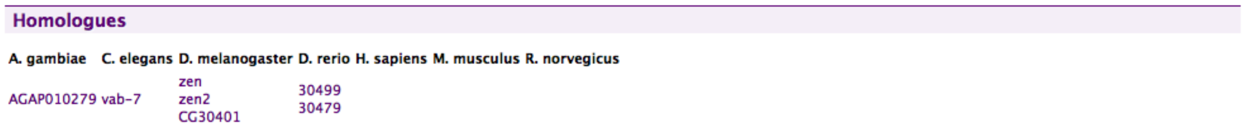


Figure 1.10: A Genome browser view in FlyMine.

Homologues

Shows a table of organism and homologous genes of homologues per organism.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.HomologueDisplayer"
    jspName="model/homologueDisplayer.jsp"
    replacesFields="homologues"
    placement="Homology"
    parameters="{'dataSets': ['TreeFam data set', 'KEGG orthologues data set']}"
    types="Gene"/>
```



Homologues

A. gambiae	C. elegans	D. melanogaster	D. rerio	H. sapiens	M. musculus	R. norvegicus
AGAP010279	vab-7	zen zen2 CG30401	30499 30479			

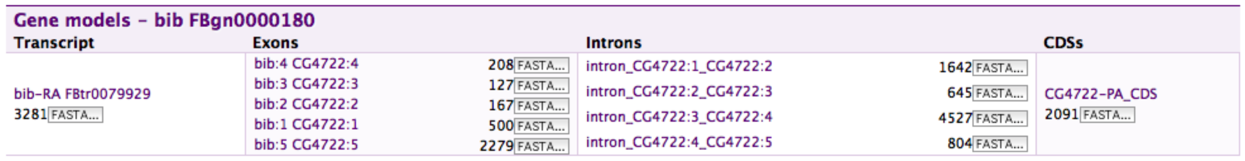
Figure 1.11: A Homologues displayer in FlyMine.

Note that FlyMine includes a specific displayer to show the twelve Drosophila species as a phylogenetic tree.

Gene structure

Displays transcripts, exons, introns, UTRs and CDSs if present in the model and for the particular organism. Can be added to report pages for any of these feature types and will find the parent gene and show all transcripts, highlighting the feature of the actual report page.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.GeneStructureDisplayer"
    jspName="model/geneStructureDisplayer.jsp"
    replacesFields="transcripts,exons,CDSs,introns,UTRs,fivePrimeUTR,threePrimeUTR"
    placement="Genomics"
    types="Gene,Transcript,Exon,Intron,UTR,CDS"/>
```



Gene models - bib FBgn0000180

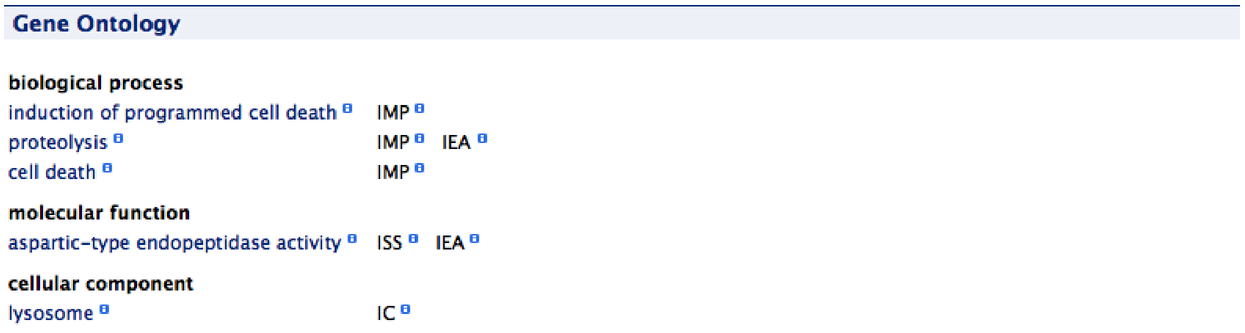
Transcript	Exons		Introns		CDSs
bib-RA FBtr0079929 3281 FASTA...	bib:4 CG4722:4	208 FASTA...	intron_CG4722:1_CG4722:2	1642 FASTA...	CG4722-PA_CDS 2091 FASTA...
	bib:3 CG4722:3	127 FASTA...	intron_CG4722:2_CG4722:3	645 FASTA...	
	bib:2 CG4722:2	167 FASTA...	intron_CG4722:3_CG4722:4	4527 FASTA...	
	bib:1 CG4722:1	500 FASTA...	intron_CG4722:4_CG4722:5	804 FASTA...	
	bib:5 CG4722:5	2279 FASTA...			

Figure 1.12: A Gene structure displayer in FlyMine.

Gene Ontology

Simple display of GO terms and evidence codes for a gene, grouped by branch in the ontology. Groups by the three main ontologies (function, process and component) so you may need to run the GO source.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.GeneOntologyDisplayer"
    jspName="model/geneOntologyDisplayer.jsp"
    replacesFields="goAnnotation,ontologyAnnotations"
    placement="Function"
    types="Gene"/>
```


Gene Ontology

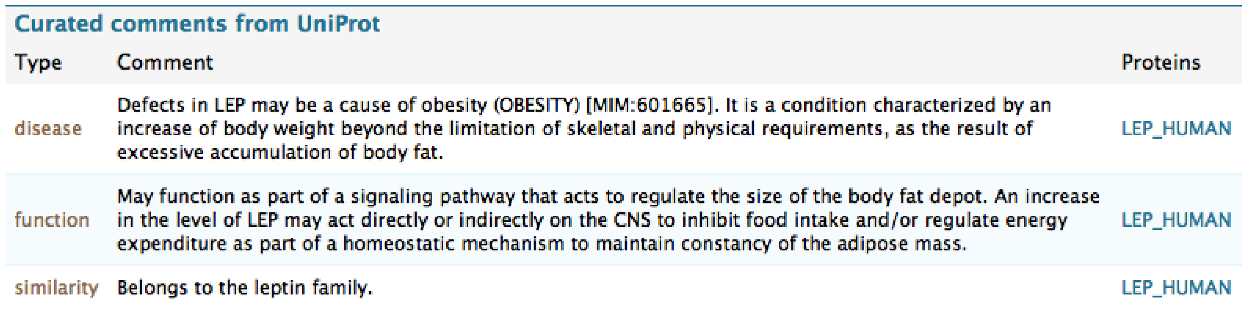
biological process		
induction of programmed cell death	IMP	
proteolysis	IMP	IEA
cell death	IMP	
molecular function		
aspartic-type endopeptidase activity	ISS	IEA
cellular component		
lysosome	IC	

Figure 1.13: A Gene ontology displayer in modMine.

UniProt comments

A clear view of curated curated comments from UniProt (SwissProt) applied to a protein, or for a gene will show comments from all proteins of the gene.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.UniProtCommentsDisplayer"
    jspName="model/uniProtCommentsDisplayer.jsp"
    replacesFields=""
    placement="summary"
    types="Gene,Protein"/>
```


Curated comments from UniProt

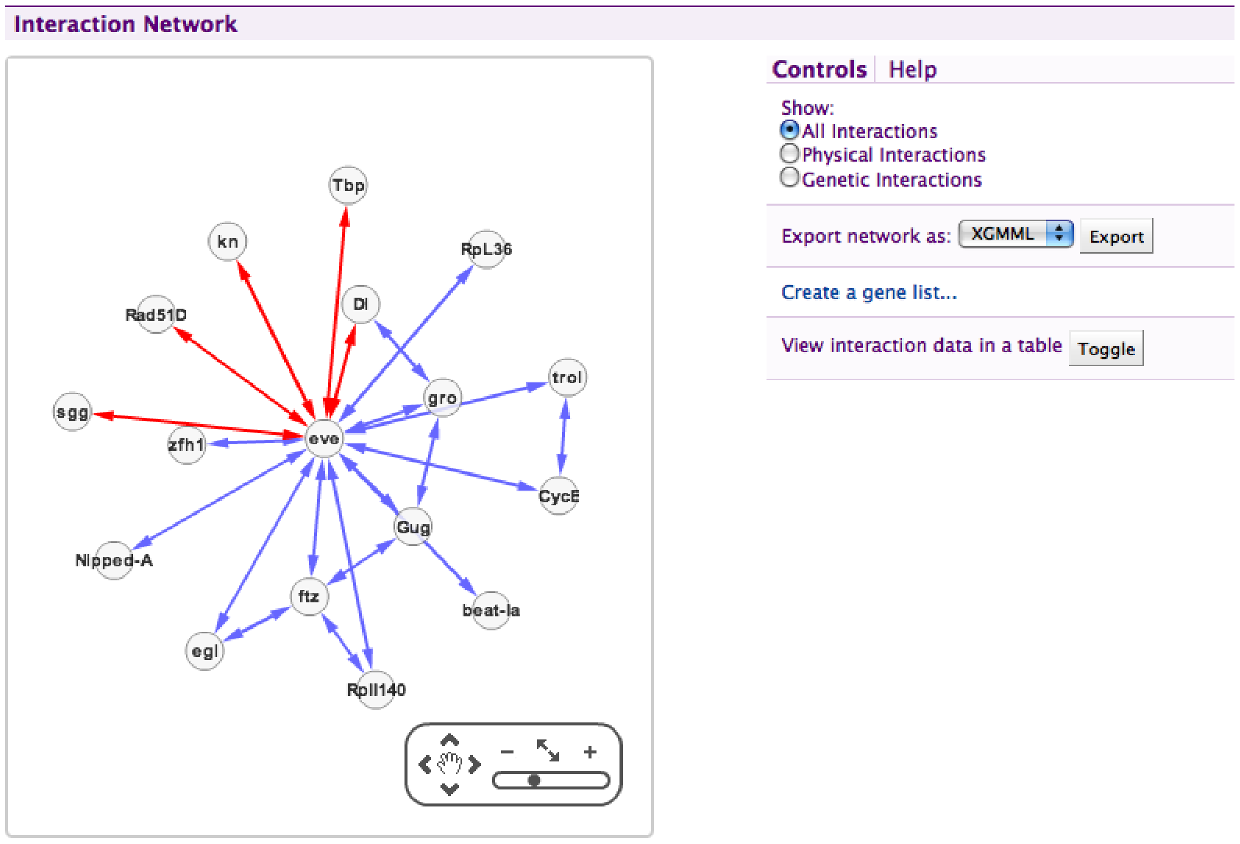
Type	Comment	Proteins
disease	Defects in LEP may be a cause of obesity (OBESITY) [MIM:601665]. It is a condition characterized by an increase of body weight beyond the limitation of skeletal and physical requirements, as the result of excessive accumulation of body fat.	LEP_HUMAN
function	May function as part of a signaling pathway that acts to regulate the size of the body fat depot. An increase in the level of LEP may act directly or indirectly on the CNS to inhibit food intake and/or regulate energy expenditure as part of a homeostatic mechanism to maintain constancy of the adipose mass.	LEP_HUMAN
similarity	Belongs to the leptin family.	LEP_HUMAN

Figure 1.14: A Uniprot curated comments displayer in metabolicMine.

Interaction network

Uses the Cytoscape Web plugin to display physical and genetics interactions. The interaction displayer links to report pages, allows creation of a gene list of the whole network and can show tabular interaction data. Read NetworkDisplayer for details.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.CytoscapeNetworkDisplayer"
    jspName="model/cytoscapeNetworkDisplayer.jsp"
    replacesFields="interactions"
    placement="Interactions"
    types="Gene,Protein"/>
```



Show:
[✓ All Interactions]
Physical Interactions
Genetic Interactions

Export network as: XGMML Export

Create a gene list...

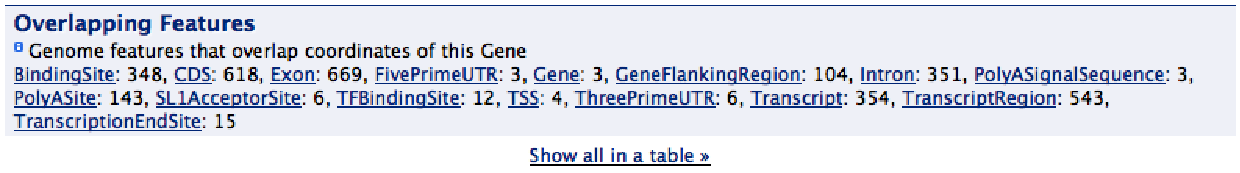
View interaction data in a table Toggle

Figure 1.15: An Interactions displayer in FlyMine.

Overlapping features

A summary view of features that overlap the chromosome location of the reported feature, if the gene structure displayer is also used it will exclude any features that are part of the same gene model - i.e. it won't report that a gene overlaps it's own exons.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.OverlappingFeaturesDisplayer"
    jspName="model/overlappingFeaturesDisplayer.jsp"
    replacesFields="overlappingFeatures"
    placement="Genomics"
    types="SequenceFeature"/>
```



Overlapping Features
Genome features that overlap coordinates of this Gene
BindingSite: 348, CDS: 618, Exon: 669, FivePrimeUTR: 3, Gene: 3, GeneFlankingRegion: 104, Intron: 351, PolyASignalSequence: 3, PolyASite: 143, SL1AcceptorSite: 6, TFBindingSite: 12, TSS: 4, ThreePrimeUTR: 6, Transcript: 354, TranscriptRegion: 543, TranscriptionEndSite: 15
Show all in a table »

Figure 1.16: An Overlapping features displayer in modMine.

Complexes - Protein interactions

Viewer displaying complex interactions. Data must be loaded from IntAct. Original Source: http://interactionviewer.org/.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.ComplexDisplayer"
                 jspName="model/complexDisplayer.jsp"
                 replacesFields=""
                 placement="summary"
                 types="Complex"/>
```

Specific Displayers

There are some displayers created for specific data sets in FlyMine, metabolicMine or modMine that may not be re-usable in other Mines but could be adapted or provide inspiration.

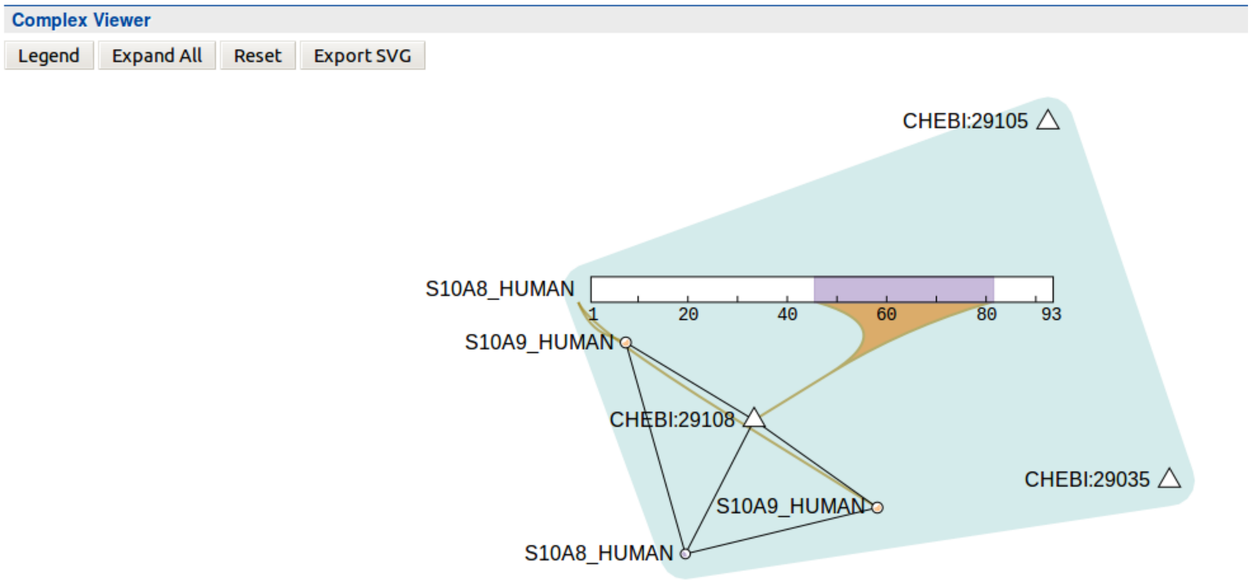
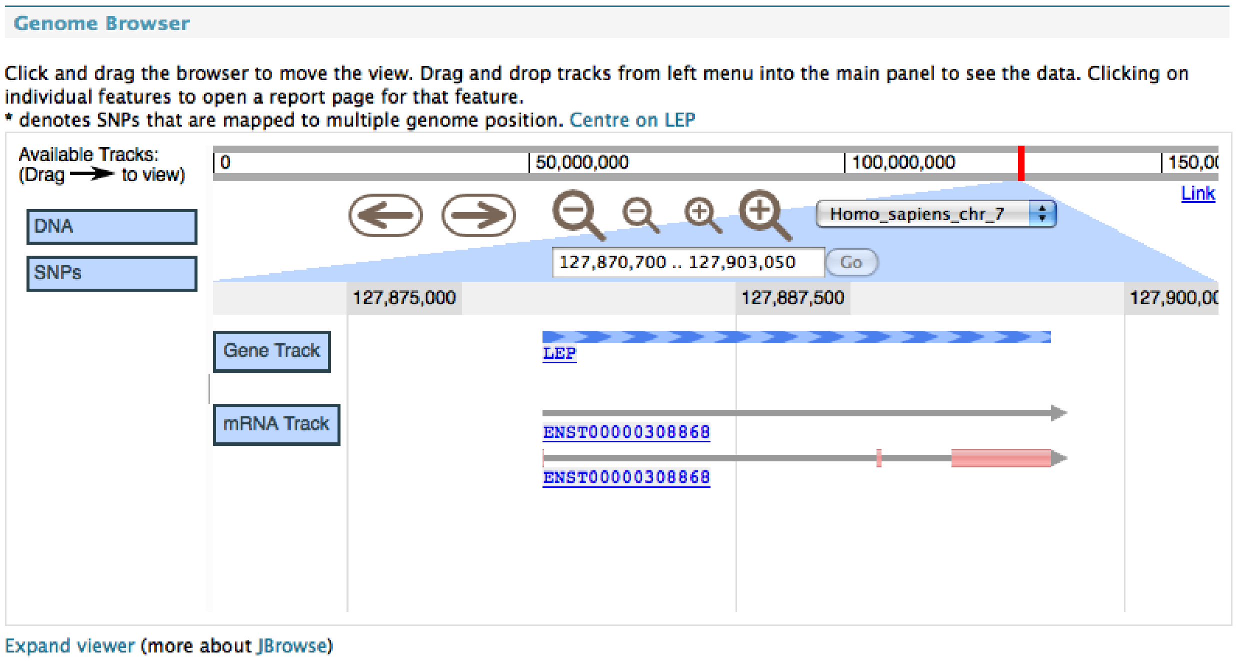


Figure 1.17: A Complex interaction displayer in HumanMine.



Click and drag the browser to move the view. Drag and drop tracks from left menu into the main panel to see the data. Clicking on individual features to open a report page for that feature.
* denotes SNPs that are mapped to multiple genome position. Centre on LEP

Expand viewer (more about JBrowse)

Figure 1.18: JBrowse genome browser in metabolicMine.

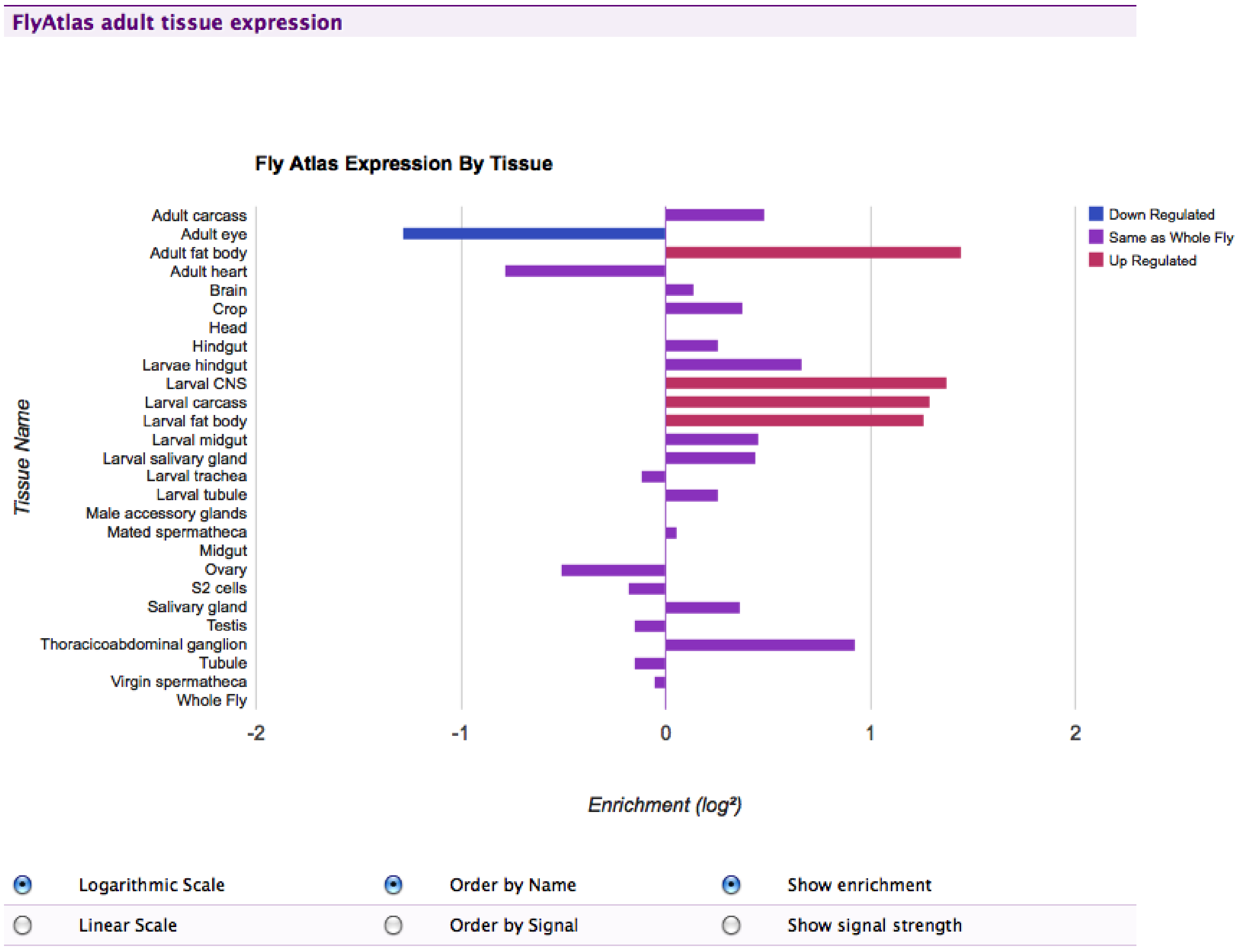


Figure 1.19: FlyAtlas gene experssion data in FlyMine, this uses the Google Data Vizualization API JavaScript library to render an interactive graph in the browser.

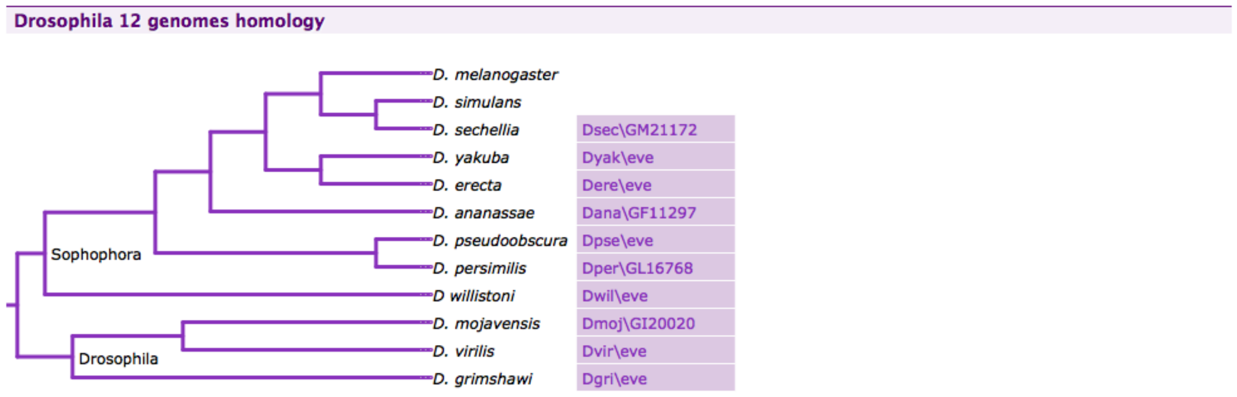
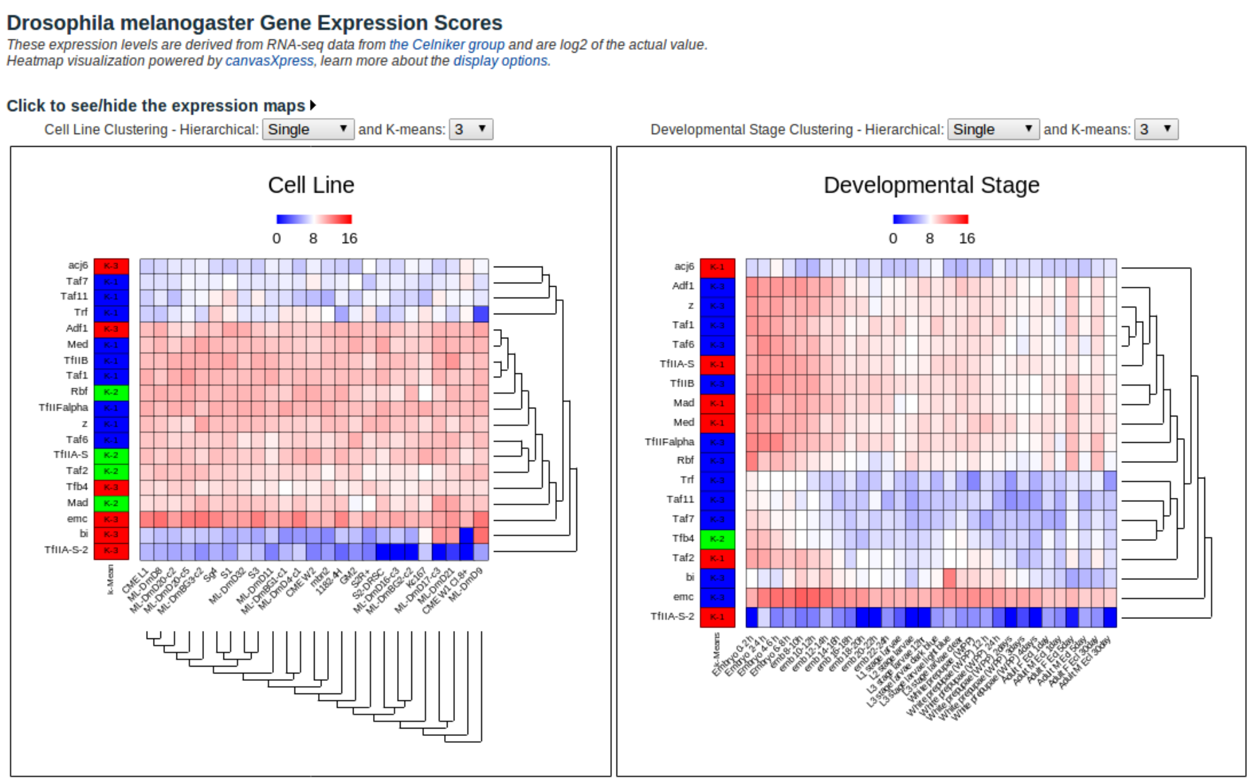


Figure 1.20: A phylogenetic tree of Drosophila species displayed using the jsPhyloSVG JavaScript library in FlyMine.



Drosophila melanogaster Gene Expression Scores

These expression levels are derived from RNA-seq data from the Celniker group and are log2 of the actual value.
Heatmap visualization powered by canvasXpress, learn more about the display options.

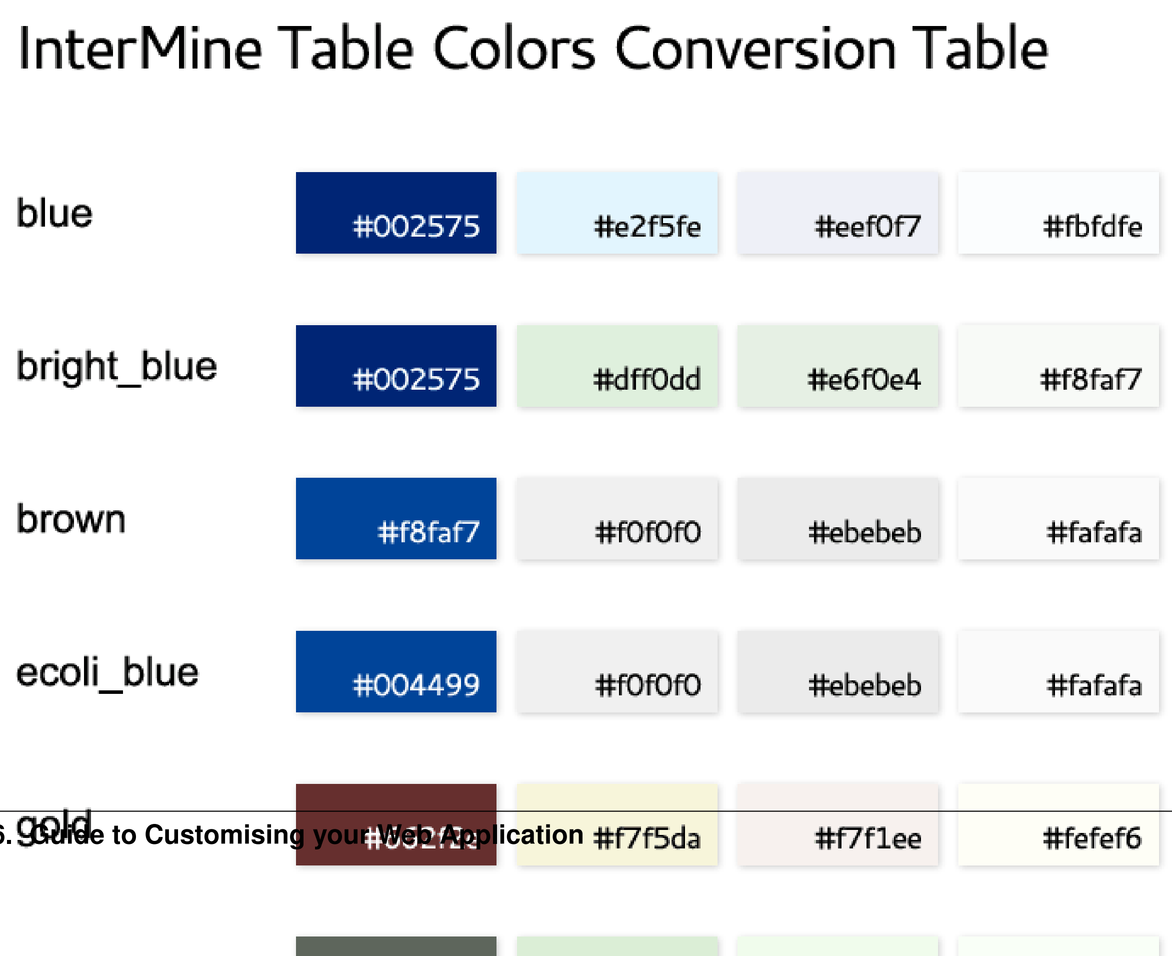
Click to see/hide the expression maps ▸

Cell Line Clustering - Hierarchical: Single ▾ and K-means: 3 ▾

Developmental Stage Clustering - Hierarchical: Single ▾ and K-means: 3 ▾

Figure 1.21: Heatmap of fly gene expression in modMine, this makes use of canvasXpress JavaScript library.

Webapp Tables CSS & HTML



InterMine Table Colors Conversion Table

blue	#002575	#e2f5fe	#eef0f7	#fbfdfe
bright_blue	#002575	#dff0dd	#e6f0e4	#f8faf7
brown	#f8faf7	#f0f0f0	#ebebeb	#fafafa
ecoli_blue	#004499	#f0f0f0	#ebebeb	#fafafa
gold	[illegible]	#f7f5da	#f7f1ee	#fefef6

Inline List

```
<div class="inline-list">
<h3>2 probeSets</h3>
<ul>
    <li>FBgn0014159, </li>
    <li><a>Complementation group F</a>, </li>
</ul>
</div>
```

CSS	Description
`div.inline-list`	wrapping the list and title in div makes it more clear what elements belong together and allow you to set a custom ID on the whole thing
`div.inline-list h3` (optional)	header 3 (see below) styling
`div.inline-list ul`	list we be displayed inline, without margins between items and without list styles (circles, squares etc.)

0 pathways

Inline List (Inactive, No Results)

```
<div class="inline-list gray">
    <h3>0 probeSets</h3>
</div>
```

CSS	Description
`div.inline-list.gray`	one can apply an 'inactive' theme by attaching a class to the top element

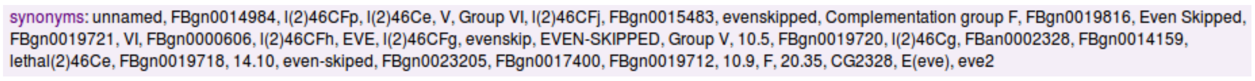
0 pathways

im:aspect:Function[x] im:summary[x] Add tags

Inline List (Tagging, Right)

```
<div class="inline-list">
    <h3><div class="right">Right positioned</div> 0 probeSets</h3>
</div>
```

CSS	Description
`div.inline-list div.right` (optional)	will float element to the right and apply appropriate colors to links; needs to go first, before any other text

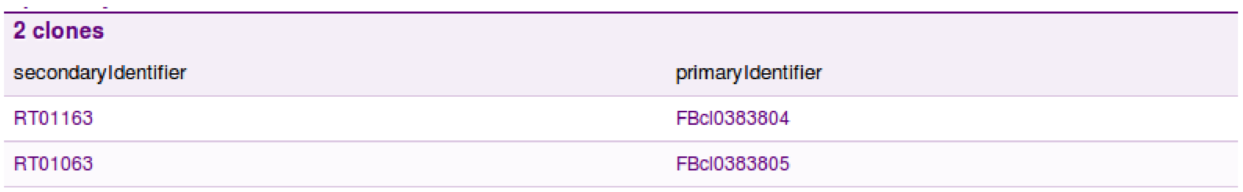

synonyms: unnamed, FBgn0014984, l(2)46CFp, l(2)46Ce, V, Group VI, l(2)46CFj, FBgn0015483, evenskipped, Complementation group F, FBgn0019816, Even Skipped, FBgn0019721, VI, FBgn0000606, l(2)46CFh, EVE, l(2)46CFg, evenskip, EVEN-SKIPPED, Group V, 10.5, FBgn0019720, l(2)46Cg, FBan0002328, FBgn0014159, lethal(2)46Ce, FBgn0019718, 14.10, even-skiped, FBgn0023205, FBgn0017400, FBgn0019712, 10.9, F, 20.35, CG2328, E(eve), eve2

'Header' Inline List

```
<div class="inline-list">
    <ul>
        <li><span class="name">synonyms</span>:</li>
        <li>FBgn0014159, </li>
        <li>Complementation group F, </li>
        <li>FBgn0015483, </li>
    </ul>
</div>
```

CSS	Description
`div.inline-list .name` (optional)	the main theme color will be applied to the element

Collection Table



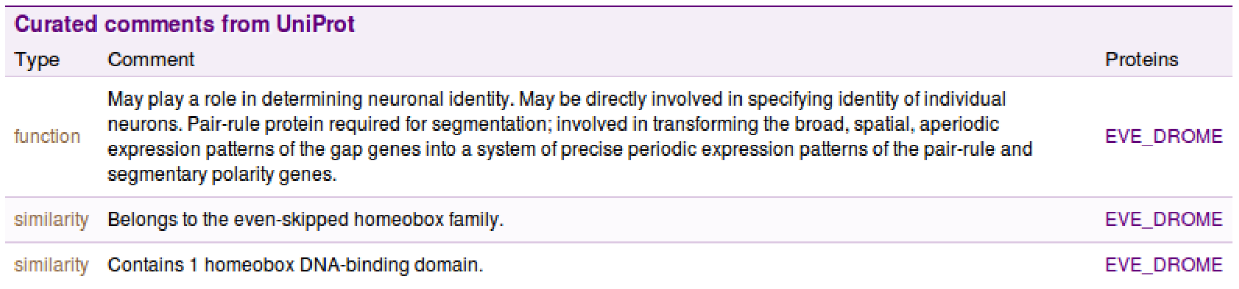
2 clones

secondaryIdentifier	primaryIdentifier
RT01163	FBcl0383804
RT01063	FBcl0383805

```
<div class="collection-table">
    <h3>1 protein</h3>
    <table>
        <thead>
            <tr><th>primaryIdentifier</th><th>primaryAccession</th></tr>
        </thead>
        <tbody>
            <tr>
                <td>EVE_DROME</td>
                <td>P06602</td>
                </tr>
                <tr>
                <td>AUTO_DROME</td>
                <td>P65</td>
            </tr>
        </tbody>
    </table>
</div>
```

CSS	Description
`div.collection-table h3`	table title will pickup theme colors much like Title (Level 3) below
`div.collection-table thead th,td`	table expects a thead element, that will apply the same background as the title
`div.collection-table.nowrap` (optional)	row columns do not wrap and are displayed inline

Note: Modern browsers will apply alternating background and border on odd row columns, the rubbish (IE) will be fixed by running jQuery on page load.



Curated comments from UniProt

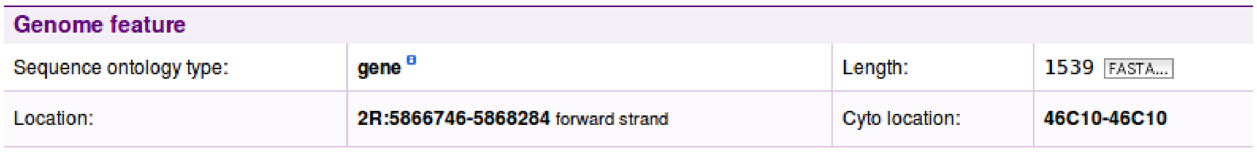
Type	Comment	Proteins
function	May play a role in determining neuronal identity. May be directly involved in specifying identity of individual neurons. Pair-rule protein required for segmentation; involved in transforming the broad, spatial, aperiodic expression patterns of the gap genes into a system of precise periodic expression patterns of the pair-rule and segmentary polarity genes.	EVE_DROME
similarity	Belongs to the even-skipped homeobox family.	EVE_DROME
similarity	Contains 1 homeobox DNA-binding domain.	EVE_DROME

Collection Table (Type Column, Text Highlight)

```
<div class="collection-table">
    <h3>1 protein</h3>
    <table>
        <thead>
            <tr><th>primaryIdentifier</th><th>primaryAccession</th></tr>
        </thead>
        <tbody>
```

```
            <tr>
                <td class="class">EVE_DROME</td>
                <td>P06602</td>
                </tr>
                <tr>
                <td class="class">AUTO_DROME</td>
                <td>P65</td>
            </tr>
        </tbody>
    </table>
</div>
```

CSS	Description
`div.collection-table table td.class`	applying a 'class' class will highlight the text in the given column



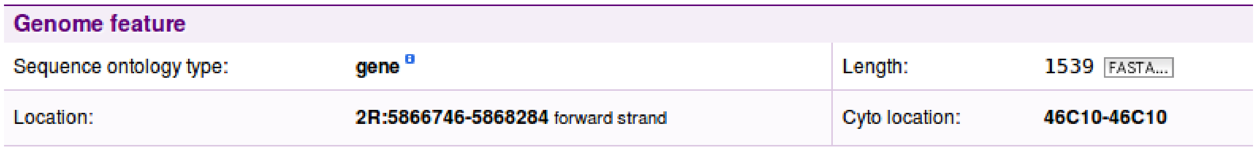
Genome feature			
Sequence ontology type:	gene	Length:	1539 FASTA...
Location:	2R:5866746-5868284 forward strand	Cyto location:	46C10-46C10

Collection Table (Vertical Column Border)

```
<div class="collection-table column-border">
    <-- ... -->
    <table>
        <-- ... -->
    </table>
</div>
```

CSS	Description
`div.collection-table.column-border`	uses a pseudoclass to apply a border between columns

Note: Modern browsers will apply alternating background and border on odd row columns, the rubbish (IE) will be fixed by running jQuery on page load.



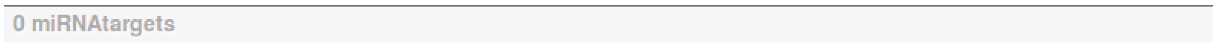
Genome feature			
Sequence ontology type:	gene	Length:	1539 FASTA...
Location:	2R:5866746-5868284 forward strand	Cyto location:	46C10-46C10

Collection Table (Vertical Column Border by 2)

```
<div class="collection-table column-border-by-2">
    <-- ... -->
    <table>
        <-- ... -->
    </table>
</div>
```

CSS	Description
`div.collection-table.column-border-by-2`	uses a pseudoclass to apply a border between every other column

Note: Modern browsers will apply alternating background and border on odd row columns, the rubbish (IE) will be fixed by running jQuery on page load.

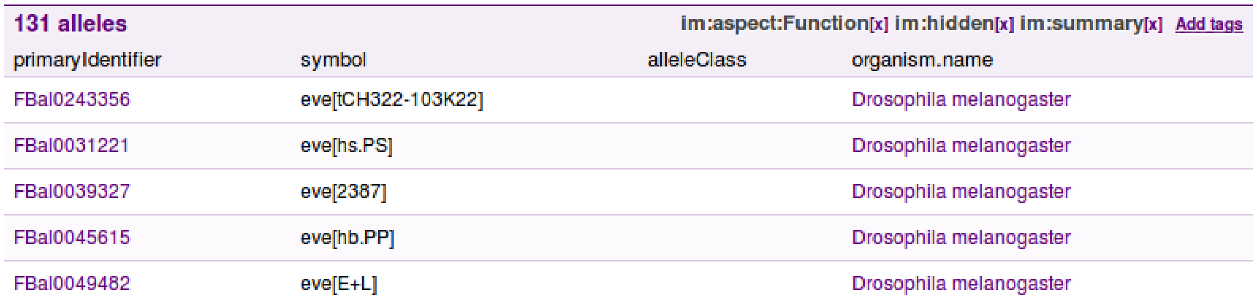


0 miRNAtargets

Collection Table (Inactive, No Results)

```
<div class="collection-table gray">
    <h3>0 genes</h3>
</div>
```

CSS	Description
`div.collection-table.gray`	one can apply an 'inactive' theme by attaching a class to the top element



131 alleles — im:aspect:Function[x] im:hidden[x] im:summary[x] Add tags

primaryIdentifier	symbol	alleleClass	organism.name
FBal0243356	eve[tCH322-103K22]		Drosophila melanogaster
FBal0031221	eve[hs.PS]		Drosophila melanogaster
FBal0039327	eve[2387]		Drosophila melanogaster
FBal0045615	eve[hb.PP]		Drosophila melanogaster
FBal0049482	eve[E+L]		Drosophila melanogaster

Collection Table (Tagging, Right)

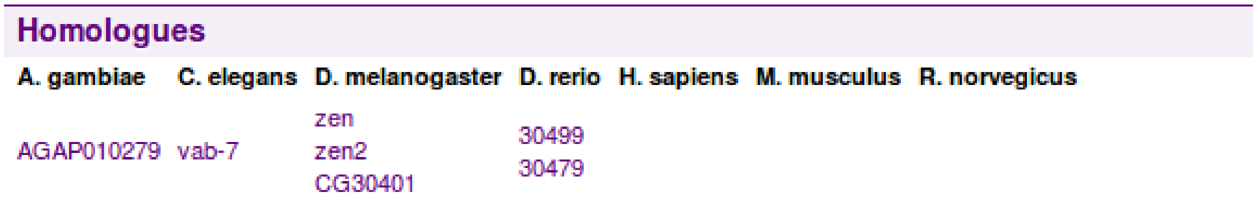
```
<div class="collection-table">
    <h3><div class="right">Right positioned</div> 0 genes</h3>
</div>
```

CSS	Description
`div.collection-table` `div.right` (optional)	will float element to the right and apply appropriate colors to links; needs to go first, before any other text

Collection Table (Persistent Table Headers)

```
<div class="collection-table persistent">
    <-- ... -->
</div>
```

CSS	Description
`div.collection-table.persistent`	will make table headers persist as you scroll within the table



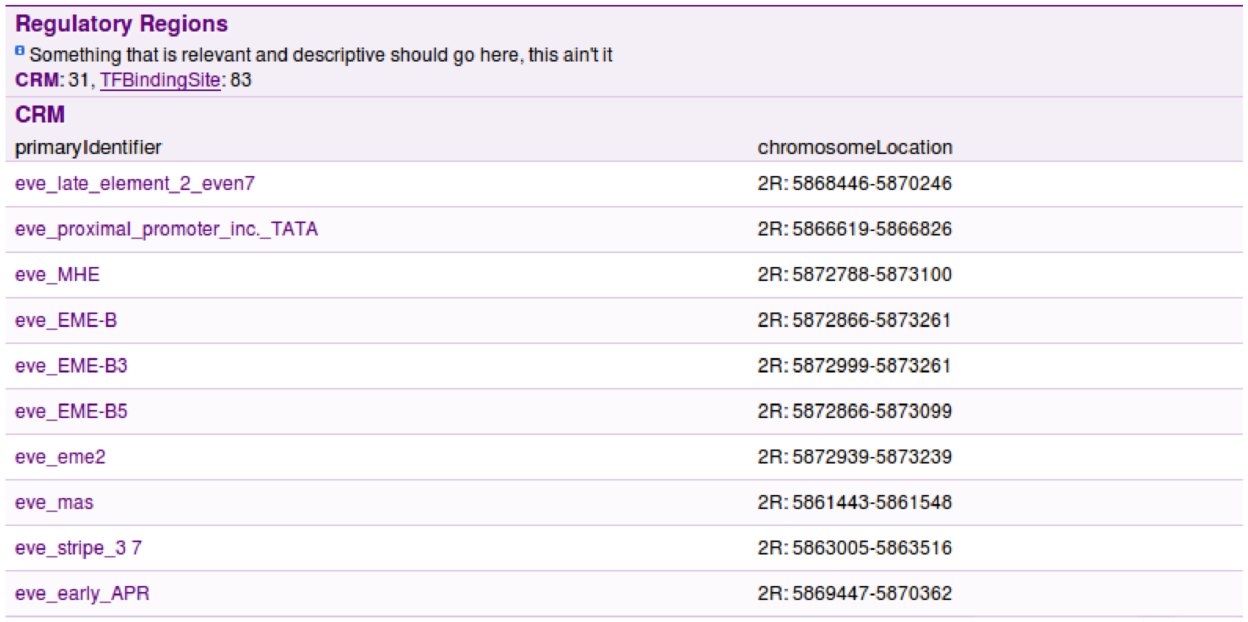
Homologues

A. gambiae	C. elegans	D. melanogaster	D. rerio	H. sapiens	M. musculus	R. norvegicus
AGAP010279	vab-7	zen zen2 CG30401	30499 30479			

Basic Table (Generic)

```
<div class="basic-table">
    <h3>Some title</h3>
    <table>
        <tr><td>Row column</td></tr>
    </table>
</div>
```

CSS	Description
`div.basic-table h3`	will apply the heading 3 style (see below)
`div.basic-table div.right` (optional)	will float element to the right and apply appropriate colors to links; needs to go first, before any other text
`div.basic-table table`	will make sure that the table is properly collapsed, has padding and does not have cellspacing
`div.basic-table.gray` (optional)	one can apply an 'inactive' theme by attaching a class to the top element



Regulatory Regions

Something that is relevant and descriptive should go here, this ain't it

CRM: 31, TFBindingSite: 83

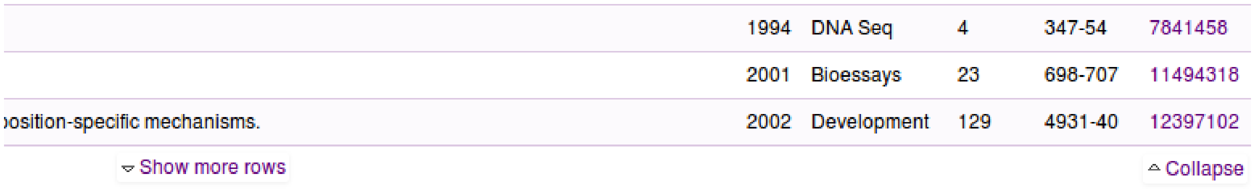
CRM

primaryIdentifier	chromosomeLocation
eve_late_element_2_even7	2R: 5868446-5870246
eve_proximal_promoter_inc._TATA	2R: 5866619-5866826
eve_MHE	2R: 5872788-5873100
eve_EME-B	2R: 5872866-5873261
eve_EME-B3	2R: 5872999-5873261
eve_EME-B5	2R: 5872866-5873099
eve_eme2	2R: 5872939-5873239
eve_mas	2R: 5861443-5861548
eve_stripe_3 7	2R: 5863005-5863516
eve_early_APR	2R: 5869447-5870362

Collection of Collection Tables

```
<div class="collection-of-collections">
    <div class="header">
        <h3>Regulatory Regions</h3>
        <p>Description</p>
        <div class="switchers">
            <a class="active">CRM</a> <a>TFBindingSite</a>
        </div>
    </div>
    <div class="collection-table">
        <-- ... -->
    </div>
    <div class="collection-table">
        <-- ... -->
    </div>
</div>
```

CSS	Description
`div.collection-of-collections`	a div wrapper for collections
`div.collection-of-collections div.header`	will apply a background color that of collection table header
`div.collection-of-collections div.header a.active` (optional)	link elements are underlined by default and switched to bold if class 'active' is applied to them



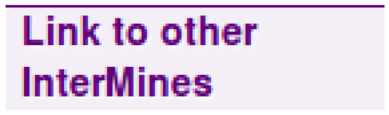
	1994	DNA Seq	4	347-54	7841458
	2001	Bioessays	23	698-707	11494318
›osition-specific mechanisms.	2002	Development	129	4931-40	12397102

Table Togglers (Less, More, Expand, Collapse, Show in table)

```
<div class="collection-table">
    <-- ... -->
    <div class="toggle">
        <a class="less">Show less</a>
        <a class="more">Show more</a>
    </div>
    <div class="show-in-table">
        <a href="#">Show all in a table</a>
    </div>
</div>
```

CSS	Description
`div.collection-table div.toggle a.more`	will create apply an expand/more button
`div.collection-table div.toggle a.less`	will create apply a collapse/less button; bear in mind that if you want to show it to the right like on report pages, it needs to go before other toggles and be floated right
`div.collection-table div.toggle a` (optional)	a generic button without any upward/downward arrows
`div.collection-table div.show-in-table a`	the appropriate color will be applied to the link contained, no more, no less (in fact, show all)

Title (Level 3)

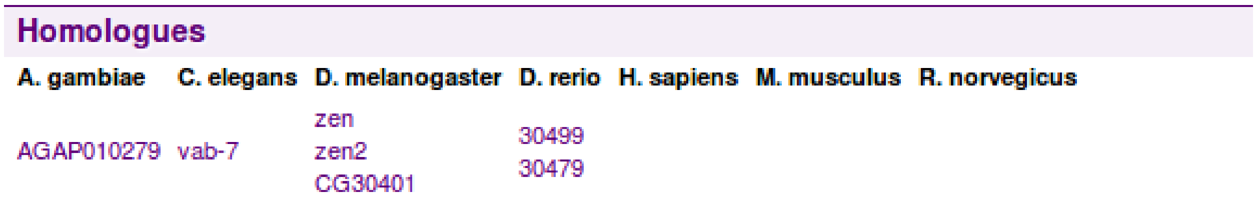


Link to other InterMines

```
<h3 class="goog">Link to other InterMines</h3>
```

CSS	Description
`h3.goog`	will pickup theme colors and apply Report Page/Google News -style colors, backgrounds, borders

Smallfont, Display one-per-line



Homologues

A. gambiae	C. elegans	D. melanogaster	D. rerio	H. sapiens	M. musculus	R. norvegicus
AGAP010279	vab-7	zen zen2 CG30401	30499 30479			

```
<table class="tiny-font">
    <tr><td class="one-line">
        <a>One</a>
        <a>Two</a>
    </td></tr>
</table>
```

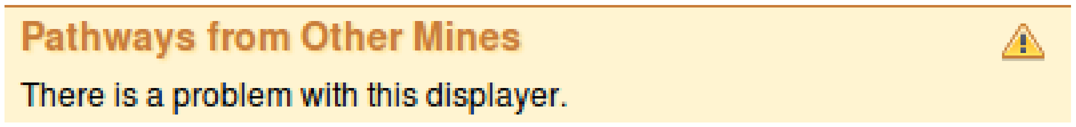
CSS	Description
`.one-line *`	applying class 'oneline' will make *all descendants* appear one per line
`.tiny-font *`	will apply 11px font size to *all descendants*

Loading spinner (AJAX)

```
<div class="loading-spinner"></div>
```

CSS	Description
`.loading-spinner`	will show an inline block positioned loading spinner gif

Table Warning



Pathways from Other Mines

There is a problem with this displayer.

```
<div class="collection-table warning">
    <-- ... -->
</div>
```

CSS	Description
`.warning`	will show a warning icon and change the color of the div to pale yellow

Report Widgets

Have been retired and made into something better. Use *Report Displayers* instead.

1.6.4 Lists

Class keys specifies unique fields for classes in the data model for the webapp.

- Fields specified in this file will be links in the results table in the webapp.
- Only objects with key fields defined can be saved in lists.

Lists page

To have lists appear on the lists page, lists a template as a SuperUser and tag the list with the *im:public* tag.

The lists are sorted by most recent first.

List upload

InterMine has a sophisticated list import system for genes. The page aims to describe how it works.

Users can paste identifiers into the list upload form; e.g. for data types "gene" it can be an identifier, symbol or name. Which key is used is determined by the class_keys file. The list upload form runs a series of queries to try to match the identifier to an object in the database.

This is also how the LOOKUP constraint works.

The list upload form runs the three queries listed below until it finds a match for the user's identifiers. It's now possible to run all three queries every time regardless of if a match was returned. You may want to configure your mine to do this if your database contains lots of identifiers that are assigned to different objects, this option would allow your users to see more options - not just the first.

Queries

Default Query First, we query for the value in key fields. For example:

```
select * from gene
where name = 'adh' OR
  symbol = 'adh' or
primaryIdentifier = 'adh' or
secondaryIdentifier = 'adh';
```

If this query returned results, that object is added to our list and we are done.

If this query didn't return any results, we move on to the next step.

Note: You can set a parameter in bag-queries.xml, matchOnFirst="false", to always run all queries.

"Bag Queries" Next we run queries listed in *bag-queries.xml*

1. looks for cross references
2. looks for synonyms

Matches returned from this query are not added to the list (if *matchesAreIssues=true*), they are displayed under the "synonyms matched" heading. Users can optionally add them to their list.

If this query didn't return any results, we move on to the next step.

Converters Next we run appropriate converter template, which are templates tagged with *im:converter*. Here is an example converter template:

```
<template name="Gene_To_Protein_Type_Converter" title="Gene to protein type converter" comment="">
        <query name="Gene_To_Protein_Type_Converter" model="genomic" view="Gene.id Gene.proteins.id"
        <constraint path="Gene.id" editable="true" description="Gene.id" op="=" value="0"/>
        </query>
</template>
```

Matches returned from this query are not added to the list, they are displayed under the "converted type" heading. Users can optionally add them to their list.

Configuration

types (classes) Add a class to *dbmodel/resources/class_keys.properties* file to get it to show up on the list upload form. To *bold* a class, tag it with *im:preferredBagType*.

organisms All organisms in your database will be displayed here. You can set the default in WebProperties.

example list The example list is set in "bag.example.identifiers" property in WebProperties.

valid delimiters The default valid delimiters are comma, space, tab or new line. You can change this value by setting the "list.upload.delimiters" property in WebProperties.

matchOnFirst Set this value in the bag-queries.xml file. Default value is TRUE. If false, all queries will always be run.

List upgrade

When you update an InterMine production database, user lists have to be updated as well. This document aims to describe this process.

Why a list "upgrade" is needed

Lists are saved in the userprofile *savedbag*, *bagvalues* tables and in the production database *osbag_int* table.

Production Database **obsbag_int table**

column	notes
bagid	unique bag id
value	intermine object id

Note: The InterMine ID is only valid per database. If the database is rebuilt, the IDs change and the information in this table becomes incorrect. The lists require an *upgrade* for them to be updated with the new, correct InterMine object IDs.

Userprofile Database **savedbag table**

column	notes
osbid	bag id
type	type of object, eg. Gene
id	id
name	name of list, provided by user
datecreated	timestamp
description	description, provided by user
userprofileid	user id
intermine_state	CURRENT, NOT_CURRENT or TO_UPGRADE

bagvalues table

column	notes
savedbagid	bag id
value	identifier originally typed in by user
extra	organism short name

Lists are saved along with the user information in the *savedbag* table. The identifiers used to create a list are also stored in the *bagvalues* table in the userprofile database. These identifiers are used to upgrade the list to internal object ids in the new production database.

To make queries fast, the list contents are stored in the production database as internal object ids. When a new production database is used, the object ids are no longer valid and need to be "upgraded".

Process

- Upgrade lists only when users log in - so we won't waste time upgrading dormant user accounts and old lists.
- Superuser lists are upgraded when the webapp is first deployed.
- The webapp knows when the lists need to be upgraded. For this purpose a *serialNumber*, identifying the production database, is generated when we build a new production db and stored in the userprofile database when we release the webapp. If the two serialNumberbs don't match, the system should upgrade the lists.

Upgrading to a new release

- When a new production db is built, all the lists have to be upgraded. Their state is set to NOT_CURRENT.
- When a user logs in, a thread will begin upgrading their saved lists to the new release - finding and writing the corresponding object ids to the production database. If there are no issues (all identifiers are resolved automatically) the state of the list is set to CURRENT.
- The user can verify the state of theirs saved bags in MyMine->Lists page.
- If there are any issues, the state of the list is set to TO_UPGRADE. These lists are shown in MyMine->List page in a separate table. The user can click on the Upgrade List link and browse in the bagUploadConfirm page where all conflicts will be displayed.
- Once the user has resolved any issues, the list can be saved clicking the button 'Upgrade a list of ...' and used for queries, etc. The state is set to CURRENT.
- If a user never logs in to a particular release, the list will not be upgraded, but can still be upgraded as normal if the log in to a later release.

Lists not current

If a list is not current:

- the user can't use it in the query/template to add list contraints
- the list is not displayed in the List->View page
- the list is displayed in MyMine->Lists page, but the column *Current* is set *Not Current*. Selecting the link, the user can resolve any issue.
- the list is not dispayed in the Lists section on the report pages

bagvalues table

The list upgrade system, needs a bagvalues table in the userprofile database, with savedbagid and value columns. This table should be generated manually, running the *load-bagvalues-table* ant task in the webapp directory. The *load-bagvalues-table* task, should create the table and load the contents of the list using the former production db, that is the same db used to create the saved lists. Every time, you re-create the userprofile database, you have to re-generate the 'bagvalues' table. In theory, you should never re-create the userprofile db, so you should run the *load-bagvalues-table* task only once.

Userprofile database

The table should be populated with one row corresponding to each row in production db osbag_int table. Each row should contain the *IntermineBag* id and the first value not empty of the primary identifier field, defined in the *class_keys* properties file.

The *bagvalues* table is updated when the user is logged in and:

- creates a new list from the result page or starting from some identifiers
- creates a new list from union, copy, intersection, subtraction operations
- add or delete some rows to/from the list
- deletes a list

When a user logs in, any lists he has created in his session become saved bags in the userprofile database, and the *bagvalues* table should be updated as well. The contents of *bagvalues* is only needed when upgrading to a new release. The thread upgrading the lists, uses the contents of bagvalues as input and, if the list upgrades with no issues:

- write values to osbag_int table
- set in the savedbag table the intermine-current to true
- update osbid.

The *intermine-current* in the table *savedbag* marks whether the bag has been upgraded. The column is generated when you create the userprofile database or when *load-bagvalues-table* has been executed.

Serial Number Overview

The list upgrade functionality uses a serialNumber that identifies the production database. The serialNumber is re-generated each time we build a new production db. On startup of the webapp, the webapp compares the production serialNumber with its own serialNumber (before stored using the production serialNumber). If the two serialNumbers match, the lists will not be updgraded; if don't, the lists are set as 'not current' and will be upgraded only when the user logs in.

There are four cases:

1. production serialNumber and userprofile serialNumber are both null ==> we don't need upgrade the list.

   Scenario: I have released the webapp but I haven't re-build the production db.

2. production serialNumber is not null but userprofile serialNumber is null ==> we need upgrade the lists.

   Scenario: I have run *build-db* in the production db and it's the first time that I release the webapp. On startup, the webapp sets *intermine_current* to false and the userprofile serialNumber value with the production serialNumber value.

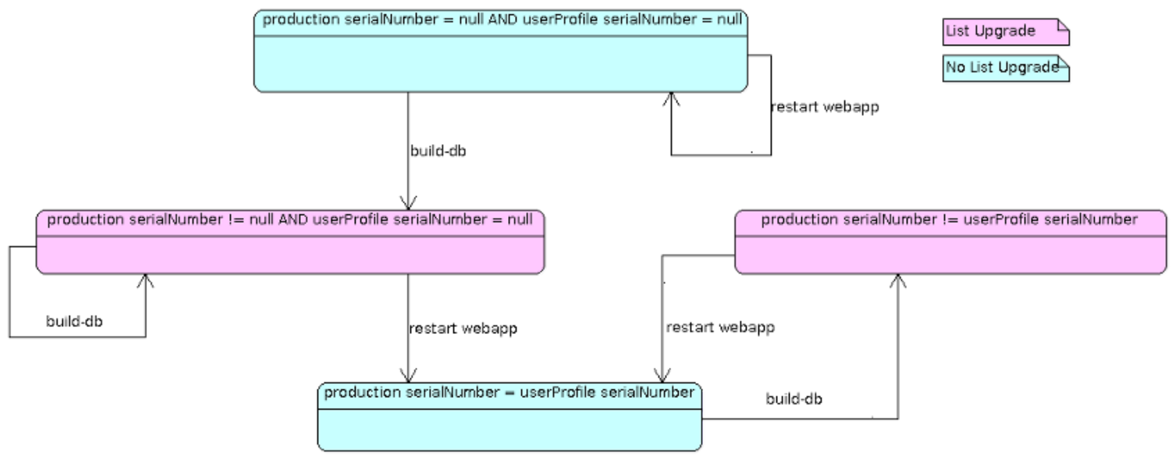
3. production serialNumber = userprofile serialNumber ==> we don't need upgrade the lists.

Scenario: we have released the webapp but we haven't changed the production db.

4. production serialNumber != userprofile serialNumber ==> we need upgrade the lists.

Scenario: we have run *build-db* in the production and a new serialNumber has been generated.

The following diagram shows the possible states. With the green, we identify the states that don't need a list upgrade, with the red those need a list upgrade.


production serialNumber = null AND userProfile serialNumber = null
List Upgrade
No List Upgrade
restart webapp
build-db
production serialNumber != null AND userProfile serialNumber = null
production serialNumber != userProfile serialNumber
build-db
restart webapp
restart webapp
production serialNumber = userProfile serialNumber
build-db

List analysis

fields displayed determined by webconfig-model.xml

export See `/webapp/query-results/export`

"Convert to a different type" Tag conversion template with *im:converter* tag. A "Conversion" template has to connect two data classes and include the id field, e.g.

```
<template name="Gene_To_Protein_Type_Converter" title="Gene to protein type converter" comment="">
        <query name="Gene_To_Protein_Type_Converter" model="genomic" view="Gene.id Gene.proteins.id"
                <constraint path="Gene.id" editable="true" description="Gene.id" op="=" value="0"/>
        </query>
</template>
```

"Orthologues" If you have orthologues loaded in your mine, you will see links in this section

"View homologues in other Mines" See *Features*

external links See *Features*

template queries Tag template with the `im:report` tag. See *Website Admin*.

widgets See: *List Widgets*

List Widgets

List Widgets Questions & Answers

Source files Source files for the List widgets client.

Using a temporary list on the fly

Requirements

1. InterMine Generic WebService Client Library from GitHub or InterMine CDN.
2. InterMine List Widgets Client Library from GitHub or InterMine CDN.

3. A mine that has the desired Enrichment Widget configured.
4. An API Access Key generated by logging in to MyMine and visiting the API Key tab, then clicking on Generate a new API key. This assumes that you do not want to automatically provide the API key as is the case of within mine embedding that can be seen for example here.

Code First require the JavaScript libraries needed to run the example. You probably have your own version of a Twitter Bootstrap compatible CSS style included on the page already.

```
<!-- dependencies -->
<script src="http://cdn.intermine.org/js/jquery/1.9.1/jquery-1.9.1.min.js"></script>
<script src="http://cdn.intermine.org/js/underscore.js/1.3.3/underscore-min.js"></script>
<script src="http://cdn.intermine.org/js/backbone.js/0.9.2/backbone-min.js"></script>

<!-- intermine -->
<script src="http://cdn.intermine.org/api"></script>
<script src="http://cdn.intermine.org/js/intermine/imjs/latest/im.js"></script>
<script src="http://cdn.intermine.org/js/intermine/widgets/latest/intermine.widgets.js"></script>
```

The next step is defining a couple of variables.

```
var root = 'http://www.flymine.org/query';
var tokn = 'U1p3r9Jb95r2Efrbu1P1CdfvKeF'; // API token
var name = 'temp-list-from-js-query'; // temporary list name
```

Now we connect with the mine through *InterMine JavaScript Library*.

```
// Service connection.
var flymine = new intermine.Service({
    'root':  root,
    'token': tokn
});
```

Then we define the query whose results will be converted into a list later on.

```
// The query herself.
var query = {
    'select': [ 'symbol', 'primaryIdentifier' ],
    'from': 'Gene',
    'where': {
        'symbol': {
            'contains': 'ze'
        }
    },
    'limit': 10
};
```

Now we call the mine converting the results of the query into a list.

```
flymine.query(query)
       .then(function madeQuery (q) {
         // q is an instance of intermine.Query.
         return q.saveAsList({'name': name}); })
       .then(function savedList (list) {
         // list is an instance of intermine.List.
         console.log(list.size); });
       .fail(function onError (error) {
         console.error("Something went wrong");});
```

Now, in the function *savedList*, we can instantiate the List Widgets client and display the result.

```
var widgets = new intermine.widgets(root + '/service/', tokn);
// A new Chart List Widget for a particular list in the target #widget.
widgets.chart('flyfish', name, '#widget');
```

The only problem with this approach is that if we make this sort of call multiple times, we will fail on the second and subsequent ocassions as we will get a WebService exception telling us that the 'temporary' list name is taken. *Thus inspect the code of the example to see how to make a call to the service to delete/reuse the list if it exists*.

Defining custom actions on widget events In a mine context, List Widgets are configured automatically to e.g. display a *Query Results* when clicking on "Create a List".

Outside of a mine context, one needs to pass in what happens when one interacts with the Widgets. You can also decide whether to show/hide either/and/or title or description of the widget (for everything else use CSS).

Clicking on an individual match (Gene, Protein etc.) in popover window:

```
var options = {
    matchCb: function(id, type) {
        window.open(mineURL + "/portal.do?class=" + type + "&externalids=" + id);
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Clicking on View results button in a popover window:

```
var options = {
    resultsCb: function(pq) {
        // ...
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Clicking on Create list button in a popover window:

```
var options = {
    listCb: function(pq) {
        // ...
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

I want to hide the title or description of a widget.

```
var options = {
    "title": false,
    "description": false
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Showing a Results Table The example below assumes that you have resolved all *Query Results* dependencies and have a PathQuery in JSON/JavaScript format that you want to display in a `#container`:

```
// Define a query as above
var pq = {from: "Gene", select: ["symbol", "organism.name"], where: {Gene: {in: "my-list"}}};
// use an instance of a Service or perhaps you already have one.
var service = new intermine.Service({'root': service, 'token': token});
```

```
// Create a new ResultsTable.
var view = new intermine.query.results.CompactView(service, pq);
// Say where to put it.
view.$el.appendTo("#container");
// Show it.
view.render();
```

List enrichment widgets statistics

Enrichment widgets are located on the list analysis page. There are a number of different types of enrichment widgets, but all list a term, a count and an associated p-value. The term can be something like a publication name or a GO term. The count is the number of times that term appears for objects in your list. The p-value is the probability that result occurs by chance, thus a lower p-value indicates greater enrichment.

Method The p-value is calculated using the Hypergeometric distribution. Four numbers are used to calculate each p-value:

```
     (M choose k) (N-M choose n-k)
P =   -----------------------------
              N choose n
```

n the number of objects in your list

N the number of objects in the reference population

k the number of objects annotated with this item in your list

M the number of objects annotated with item in the reference population

Apache library - Hypergeometric Distribution

Multiple Test Correction When multiple tests (statistical inferences)are run in parallel, the probability of false positive (Type I) errors increases. To address this issue, many multiple test corrections have been developed to take into account the number of tests being carried out and to correct the p-values accordingly. Enrichment widgets have three different multiple test corrections: Bonferroni, Holm-Bonferroni, and Benjamini Hochberg.

In enrichment widgets the number of “tests run” is the number of terms associated with objects in the “reference list”. Please Note, in earlier versions of InterMine (0.95 and below) the number of “tests run” was the number of terms associated with objects in the “query list”. This change has made the multiple test correction more rigorous, and will reduce the occurrence of spuriously low p-values.

Each enrichment widget has four test correction options:

None No test correction performed, these are the raw results. These p-values will be lower (more significant) than if test correction was applied.

Bonferroni Bonferroni is the simplest and most conservative method of multiple test correction. The number of tests run (the number of terms associated with objects in the reference list) is multiplied by the un-corrected p-value of each term to give the corrected p-value.

Holm-Bonferroni

```
Adjusted p-value = p-value x (number of tests - rank)
```

Benjamini Hochberg This correction is the less stringent than the Bonferroni, and therefore tolerates more false positives.

```
Adjusted p-value = p-value x (number of tests/rank)
```

1. The p-values of each gene are ranked from the smallest to largest.
2. The p-value is multiplied by the total number of tests divided by its rank.

Gene length correction The probability of a given set of genes being hit in a ChIP experiment is amongst other things proportional to their length – very long genes are much more likely to be randomly hit than very short genes are. This is an issue for some widgets – for example, if a given GO term (such as gene expression regulation) is associated with very long genes in general, these will be much more likely to be hit in a ChIP experiment than the ones belonging to a GO term with very short genes on average. The p-values should be scaled accordingly to take this into account. There are a number of different implementations of corrections, we have choosen the simplest one. The algorithm was developed by Taher and Ovcharenko (2009) for correcting GO enrichment. Corrected probability of observing a given GO term is equal to the original GO probability times the correction coefficient CCGO defined for each GO term.

```
Adjusted P = P x CCGO
```

where the correction coefficient CCGO is calculated as:

```
           LGO/LWH
CCGO = ----------------
           NGO/NWG
```

LGO Average gene length of genes associated with a GO term

LWG Average length of the genes in the whole genome

NGO Number of genes in the genome associated with this GO term

NWG Total number of genes in the whole genome.

Note: The relevant InterMine source.

Reference population The reference population is by default the collection of **all the genes with annotation** for the given organism. This can be changed to any available list of genes.

References

GOstat: Find statistically overrepresented Gene Ontologies within a group of genes
Beissbarth T, Speed TP.
Bioinformatics. 6.2004; 20(9): 1464-1465.
PubMed id: 14962934

GO::TermFinder–open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes
Boyle EI, Weng S, Gollub J, Jin H, Botstein D, Cherry JM, Sherlock G.
Bioinformatics. 2004 Dec 12;20(18):3710-5. Epub 2004 Aug 5.
PubMed id: 15297299

Controlling the false discovery rate: a practical and powerful approach to multiple testing

Benjamini, Yoav; Hochberg, Yosef

Journal of the Royal Statistical Society. 1995, Series B (Methodological) 57 (1): 289–300.

Augmentation Procedures for Control of the Generalized Family-Wise Error Rate and Tail Probabilities for the Proportion of False Positives

van der Laan, Mark J.; Dudoit, Sandrine; and Pollard, Katherine S.

Statistical Applications in Genetics and Molecular Biology: Vol. 3 : Iss. 1, Article 15, 2004.

What's wrong with Bonferroni adjustments

Perneger, TV.

BMJ Publishing Group. 1998;316:1236.

Variable locus length in the human genome leads to ascertainment bias in functional inference for non-coding elements

Taher, L. and Ovcharenko, I. (2009), *Bioinformatics <http://bioinformatics.oxfordjournals.org/content/25/5/578>* Vol. : Iss. 5: 578–584.

Note: You can read more about **Hypergeometric Distribution** at Simple Interactive Statistical Analysis or Wolfram MathWorld. **Bonferroni Correction** is discussed in this Wolfram MathWorld article.

There are several list widgets (widgets from now on) available on the InterMine list analysis page, and they are configured in *Data and Widget Configuration*.

There are three categories of widgets:

table displays the counts from the list for the collection specified

graph displays a chart based on a dataset you specify

enrichment displays the p-values of objects that appear in your list

To add a widget to your mine:

1. add config to your `webconfig-model.xml` file
2. re-release your webapp
3. view widget in a list analysis page

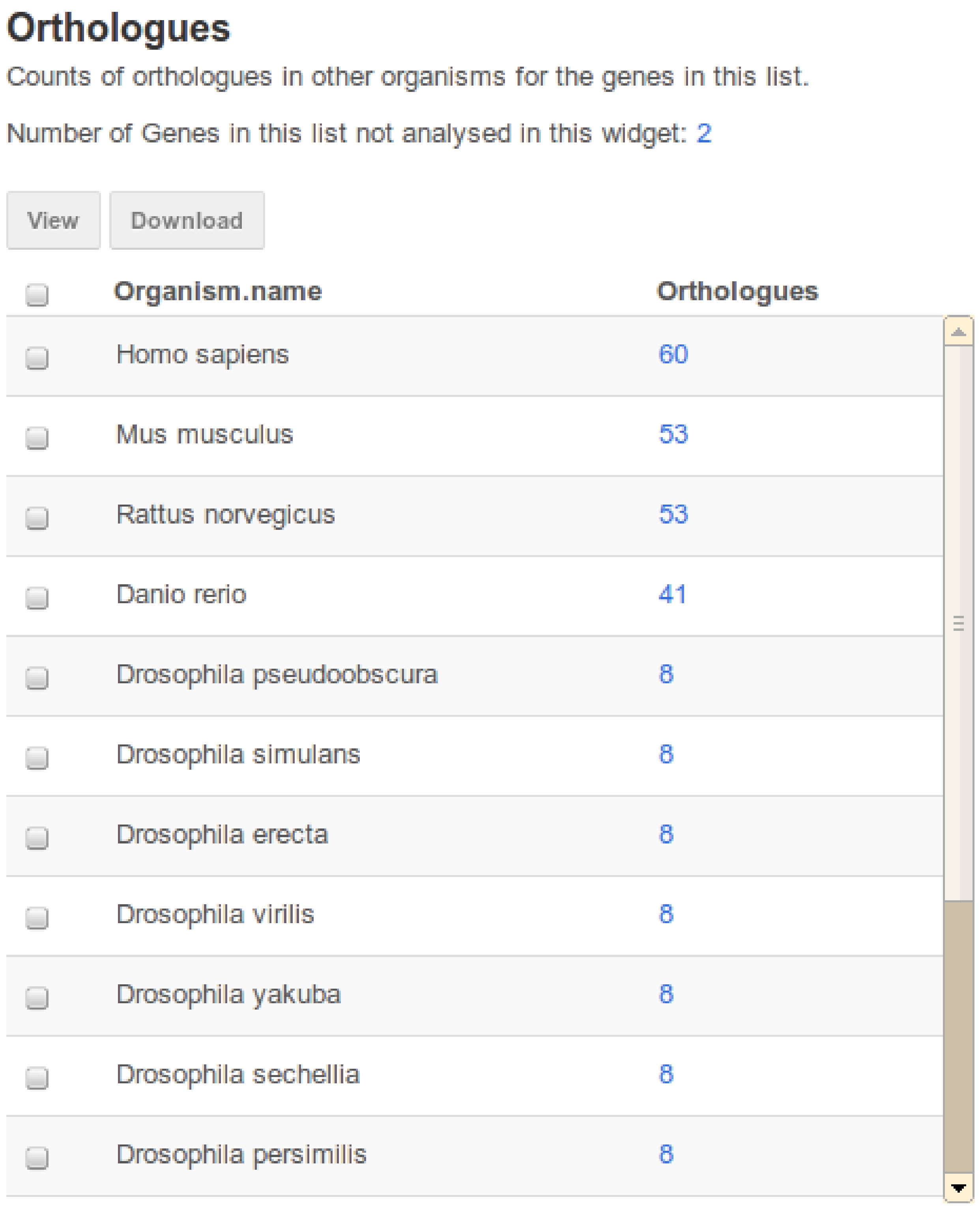
Below are the details on how to configure each widget type.

Note: Please read the documentation carefully and check your config file for typos. Most attributes are case sensitive. When the webapp is released, the config is validated and any errors displayed in the home page.

Configuration

Table widgets Table widgets display objects and the counts of related objects in your list.

An example table widget of Orthologues in FlyMine.



Orthologues

Counts of orthologues in other organisms for the genes in this list.

Number of Genes in this list not analysed in this widget: 2

Organism.name	Orthologues
Homo sapiens	60
Mus musculus	53
Rattus norvegicus	53
Danio rerio	41
Drosophila pseudoobscura	8
Drosophila simulans	8
Drosophila erecta	8
Drosophila virilis	8
Drosophila yakuba	8
Drosophila sechellia	8
Drosophila persimilis	8

attribute	purpose	example
`id`	unique id used by javascript only. Spaces not allowed.	`unique_id`
`pathStrings`	which collection to use in the widget	`Gene.homologues[type=orthologue].homologue.orga`
`exportField`	which field from the objects in your list to export	`primaryIdentifier`
`typeClass`	types of lists that should display this widget. Use the simple class name	`Gene`

The following are optional attributes:

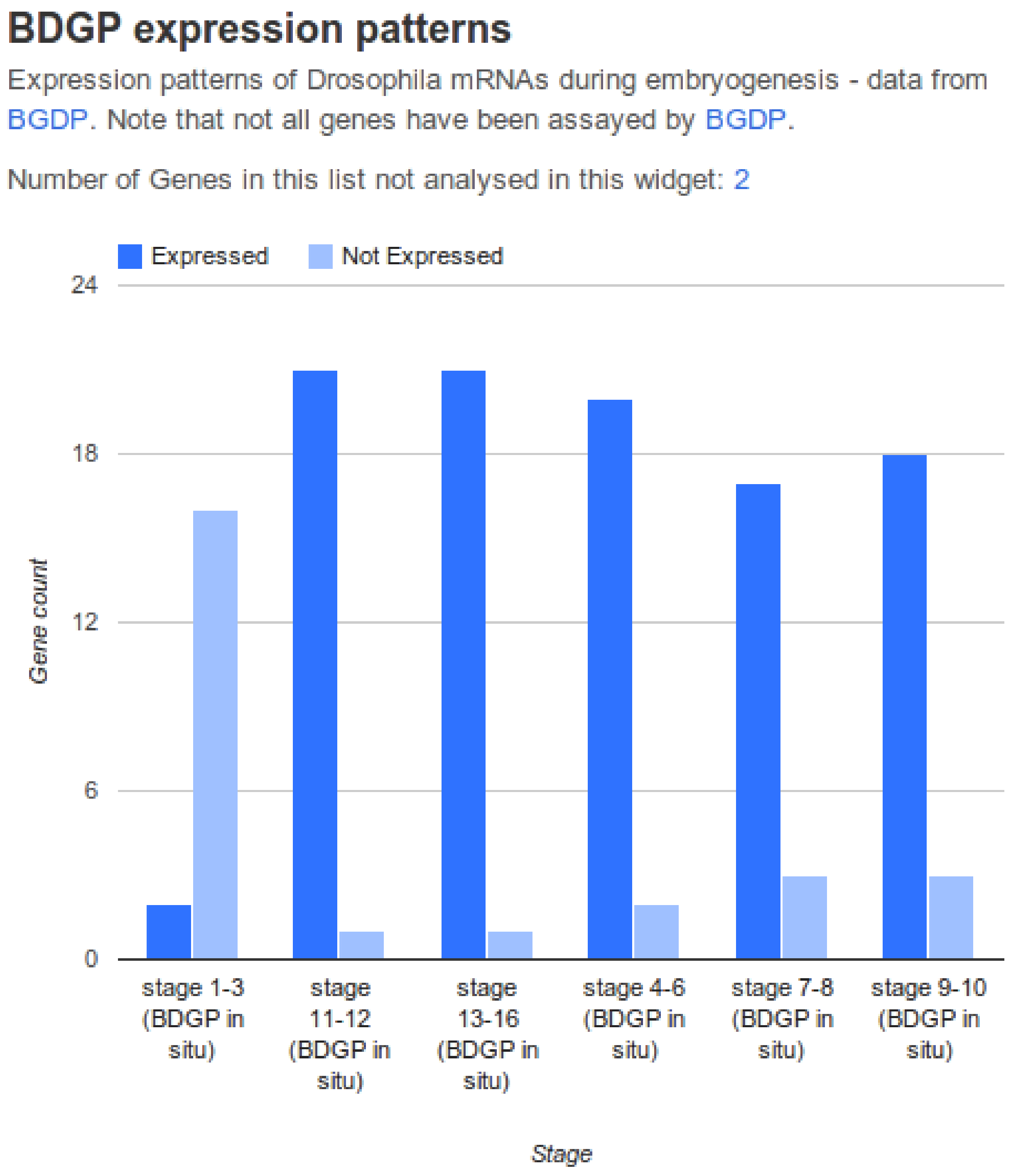
attribute	purpose	example
`title`	appears at the top of the widget	`Orthologues`
`description`	description of the widget	`Counts of orthologues`
`displayFields`	which fields from the objects in the collection (in the above example, `Gene.proteins`) to display, eg. `primaryAccession`	`name`
`columnTitle`	heading for the “count” column	`Orthologues`
`externalLink`	link displayed next to first column, identifier will be appended to link	
`externalLinkLabel`	label for external link	
`views`	path fields display in the query running when the user clicks on the widget	`symbol`

Graph/Chart widgets Graph widgets display datasets in graphical format.

attribute	purpose	example
id	unique id used by javascript only. Spaces not allowed.	`unique_id`
graphType	which type of chart to render	`ColumnChart`,‘‘BarChart‘‘ or `PieChart`
startClass	it’s the root class for all the paths specified in the configuration [6].	`Gene`
typeClass	type of lists that should display this widget. Use the simple class name.	`Gene`
categoryPath	Must be attribute. We can specify the subclass using the syntax `path[subclass type]`	`mRNAExpressionResults.stageR`
seriesPath	the series path. This has to be an attribute. We can specify the subclass using the syntax `path[subclass type]`	`mRNAExpressionResults.express`
seriesValues	the values of different series. Case sensitive. You can specify boolean values	`true,false` or `Up,Down`
seriesLabels	the labels displayed on the graphs to distinguish inside a category the different series	`Expressed,Not Expressed` or `Up,Down`
views	attributes paths displayed when the user clicks an area on the graph	`name,organism.name`

Warning: You can specify **only one** class in `typeClass`. If you need another type, you have to define a new widget.

[6] All the paths set, will be built starting from that. Specify only the simple name (e.g. `Gene`). You can choose to set the bag type class or the root class associated to the category path.



BDGP expression patterns

Expression patterns of Drosophila mRNAs during embryogenesis - data from BGDP. Note that not all genes have been assayed by BGDP.

Number of Genes in this list not analysed in this widget: 2

Figure 1.22: An example chart widget of BDGP Expression Patterns in FlyMine.

The following are optional attributes:

attribute	purpose	example
`title`	appears at the top of the widget	`BDGP expression patterns`
`description`	description of the widget	`Expression patterns`
`domainLabel`	Label displayed on x-axis in the ColumnChart (on y-axis in the BarChart)	`Stage`
`rangeLabel`	Label displayed on y-axis in the ColumnChart (on x-axis in the a BarChart)	`Gene count`
`filterLabel`	label for filter form field	`Organism`
`filters`	the values for the filter, set in the dropdown [7].	`All,KEGG pathways,Reactome data`
`listPath`	the path used to build the bag constraint [8].	`FlyAtlasResult.material`
`constraints`	separated by comma, case sensitive, must be attributes, operator can be = or != [9]	`organism.name=[Organism]` [10]

Note: The graphs use Google Visualitation API.

Enrichment widgets Enrichment widgets calculate p-values representing the probability annotation occurred by chance. See *List enrichment widgets statistics* for more information on how the p-value is calculated.

attribute	purpose	example
`id`	unique id used by JavaScript only. Spaces not allowed.	`unique_id`
`startClass`	Root class for all the paths specified in the configuration. Use simple name (e.g. Gene)	`Gene`
`startClassDisplay`	Field displayed when user clicks on the widget on 'Matches' column	`primaryIdentifier`
`typeClass`	Type of lists that should display this widget. Use the simple class name.	`Gene`
`enrich`	Field to be enriched, displayed in the widget in the firts column [11].	`goAnnotation.ontologyTerm.pare`
`views`	attributes paths displayed when the user clicks on *View results* button [6].	`symbol,organism.name`

Warning: You can specify **only one** class in `typeClass`. If you need another type, you have to define a new widget.

The following are optional attributes:

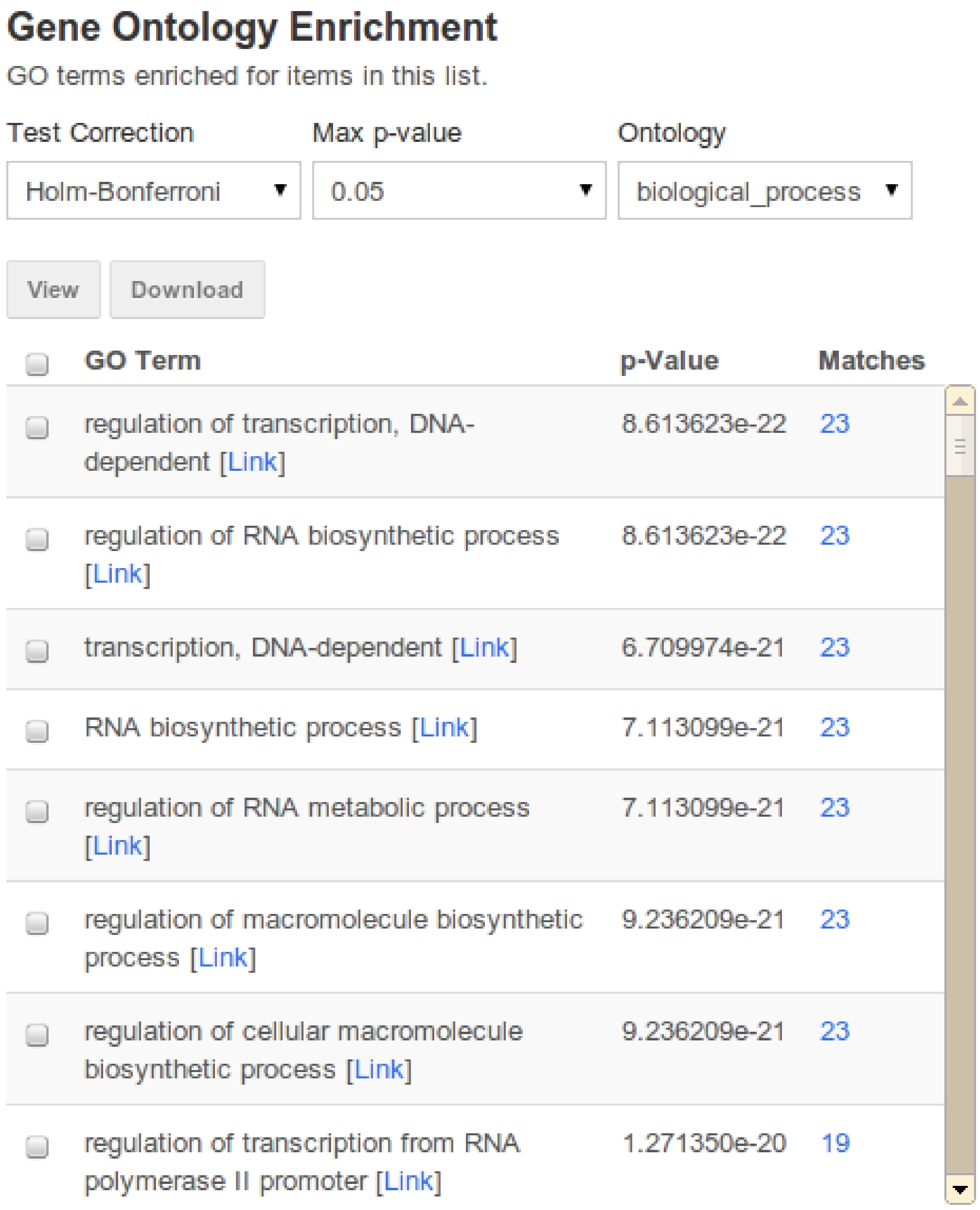
[7] We can use static values or a grammar to specify the values contained in the list. The default value in general is the first value set in the 'filters' attribute or the first value returned by the query. With static values, you can add 'All' meaning no filter applied.

[8] Optional if the `startClass` contains the bag type class.

[9] For the values we can use static values or the selected filter value using the syntax: `path constraint = [filter identifier]`.

[10] organism's name matching with the value selected in the filter with filterLabel 'Organism'

[11] You have to specify only one field. Specify the subclass using the syntax `path[subclass type]`.



Gene Ontology Enrichment

GO terms enriched for items in this list.

Test Correction: Holm-Bonferroni

Max p-value: 0.05

Ontology: biological_process

GO Term	p-Value	Matches
regulation of transcription, DNA-dependent [Link]	8.613623e-22	23
regulation of RNA biosynthetic process [Link]	8.613623e-22	23
transcription, DNA-dependent [Link]	6.709974e-21	23
RNA biosynthetic process [Link]	7.113099e-21	23
regulation of RNA metabolic process [Link]	7.113099e-21	23
regulation of macromolecule biosynthetic process [Link]	9.236209e-21	23
regulation of cellular macromolecule biosynthetic process [Link]	9.236209e-21	23
regulation of transcription from RNA polymerase II promoter [Link]	1.271350e-20	19

Figure 1.23: An example enrichment widget of Gene Ontology in FlyMine.

attribute	purpose	example
`title`	appears at the top of the widget	`Gene Ontology Enrichment`
`description`	description of the widget	`GO terms enriched.`
`label`	heading for the column	`GO Term`
`externalLink`	link displayed next to first column	`googie`
`filters`	extra filters to add to the display [12]	`organism.name=[list]`
`filterLabel`	label for filter form field	`Ontology`
`enrichIdentifier`	identifier for the row displayed, if not specified, enrich field used [13].	`goAnnotation.ontologyTerm.ide`
`constraints`	constraints separated by comma. The paths have to be attributes. The operator can be = or `!=` [14].	`organism.name=[list]`
`constraintsForView`	constraints separated by comma used for building the query executed when the user clicks on the widget on 'Matches' column	`results.expressed = true`
`correctionCoefficient`	set to org.intermine.bio.web.widget.GeneLenghtCorrectionCoefficient to normalize by gene length	

Examples See other mines' config files for more examples, eg:

- FlyMine's webconfig-model.xml
- HumanMine's webconfig-model.xml

Background population In the enrichement widgets, you can change the reference population. The reference population is specific for widget, list and user. If you are logged you can save your preference selecting the checkbox 'Save your preference'. The background population selected should include all items contained in the list.

Gene length correction coefficient Depending on the type of experiment your data comes from, it is sometimes necessary to normalize by gene length in order to get the correct p-values. If your data comes from a genome-wide binding experiment such as ChIP-seq or DamID, binding intervals are more likely to be associated with longer genes than shorter ones, and you should therefore normalize by gene length. This is not the case for experiments such as gene expression studies, where gene length does not play a role in the likelihood that a particular set of genes will be overrepresented in the list. If you want normalize by gene length, add the attribute correctionCoefficient set to 'org.intermine.bio.web.widget.GeneLenghtCorrectionCoefficient'. The gene length correction coefficient is applicable only for lists containing genes with a length, so for a list of genes do not have a length the option is not shown. If a list contains some genes without a length these genes will be discarded.

Export Values The exported file from enrichment widgets includes the enrichment identifier as the fourth column. It is contextual to the startClass attribute in the configuration. For example, an enrichment widget for publications would return the PubMedID field, where a GO enrichment widget would return the GO Term field.

Displaying widgets

JavaScript

[12] Use static values or a grammar to specify the values contained in the list. The default value in general is the first value set in the 'filters' attribute or the first value returned by the query. With static values, you can add 'All' meaning no filter applied.

[13] Specify only one. This has to be an attribute. Used in the results table. Specify the subclass using the syntax `path[subclass type]`.

[14] Case sensitive. For the values we can use: static values the selected filter value using the syntax: `path contraint = [filter identifier]` only the value contained in the list.

Widget service Create a new Widgets instance pointing to a service:

```
var widgets = new intermine.widgets("http://beta.flymine.org/beta/service/");
```

Choose a widget Choose which widget(s) you want to load:

```
// Load all Widgets:
widgets.all('Gene', 'myList', '#all-widgets');
// Load a specific Chart Widget:
widgets.chart('flyfish', 'myList', '#widget-1');
// Load a specific Enrichment Widget:
widgets.enrichment('pathway_enrichment', 'myList', '#widget-2');
// Load a specific Table Widget:
widgets.table('interactions', 'myList', '#widget-3');
```

CSS

Note: Widgets are using Twitter Bootstrap CSS framework.

Embedding mine widgets on a custom page Following is a documentation describing how to embed widgets not in a mine context.

1. Open up a document in your text editor.
2. Use the *InterMine JavaScript API Loader* that always gives you the latest version of the widgets. In the `<head>` element of the page, add the following line:

   ```
   <script src="http://cdn.intermine.org/api"></script>
   ```

3. Load the Widget Service:

   ```
   <script type="text/javascript">
       intermine.load('widgets', function() {
           var Widgets = new intermine.widgets('http://beta.flymine.org/beta/service/');
       });
   </script>
   ```

   `intermine.load` represents a block of code that loads the widgets by pointing them to a specific mine.

4. Use the widget web service to view which widgets are available on the mine, eg: *http://beta.flymine.org/beta/service/widgets/*
5. See which lists are available in the mine: *http://beta.flymine.org/beta/service/lists*
6. Add a widget (from the list in the previous step) to JavaScript. So within the `intermine.load` block, after creating the `Widgets` instance, do this:

   ```
   // Load all Widgets:
   Widgets.all('Gene', 'myList', '#all-widgets');
   // Load a specific Chart Widget:
   Widgets.chart('flyfish', 'myList', '#widget-1');
   // Load a specific Enrichment Widget:
   Widgets.enrichment('pathway_enrichment', 'myList', '#widget-2');
   // Load a specific Table Widget:
   Widgets.table('interactions', 'myList', '#widget-3');
   ```

   Where the *first parameter*‘ passed is either type of object or name of widget to load. The *second* is the name of list (public list) to access and *third* is an element on the page where your widgets will appear. This element needs to obviously exist on the page first. A common one is a div that would look like this: `<div id="all-widgets"></div>`.

7. Add HTML, eg:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>test</title>
    <script src="http://cdn.intermine.org/api"></script>
    <script type="text/javascript">
        intermine.load('widgets', function() {
            var Widgets = new intermine.widgets('http://beta.flymine.org/beta/service/');
            // Load all Widgets:
            Widgets.all('Gene', 'myList', '#all-widgets');
        });
    </script>
</head>

<body>
    <!-- DIV goes here -->
    <div class="widget" id="all-widgets">
</body>
</html>
```

8. You will have noticed that the widgets either pickup a style (CSS) from your HTML page, or they appear unstyled. To style them, you can use a variant of Twitter Bootstrap.

1.6.5 Template Queries

There are several processes run after the data loading is completed, one of which the objectstore summarisation. This step counts the number of objects of particular classes, identifies any empty references/collections and collects values to be appear in dropdowns in the query builder and templates. The summarisation process also constructs the indexes needed for "type-ahead" autocompletion, this is configured by adding entries to the `/database/database-building/post-processing/objectstore-summary-properties` file.

Dropdowns

Some fields have only a few different values, and are represented as dropdowns on forms so that users may see all possible values. You can set the maximum number of values to display, the default is 200.

To update a template query's dropdowns to only legal values, navigate to the templates page in "my mine" and click on the "summarise" link.

- All editable constraints are dropped, non-editable constraints are kept
- Valid values (summaries) for dropdowns are recalculated

Also, if your database has tables that should be ignored, you can set this too:

```
# in objectstoresummary.config.properties
ignore.counts=org.intermine.model.bio.GOAnnotation.withText org.intermine.model.bio.Location.subject
```

Organism

To populate the organism dropdown, include the *Organisms* data source in your build. Many of the tools available in InterMine assume this source will be loaded and expect a populated organism table.

Auto-completion

Fields in template queries and the QueryBuilder can have type-ahead autocompletion to assist in selecting valid terms. As you start to type, possible matches are fetched from the database; the text you have typed can match anywhere within the terms and multiple words can be matched. This is particularly useful for ontology terms or protein domain names.

You can set up autocompletion by completing these steps:

1. Add the postprocess to your *MINE_NAME/project.xml* file.

```
<post-processing>
  ...
  <post-process name="create-autocomplete-index"/>
</post-processing>
```

2. Then run this command:

```
# run postprocess
~/git/flymine $ ./gradlew postprocess -Pprocess=create-autocomplete-index --stacktrace
```

This process will add all fields set in this properties file to the autocompletion index.

Now, when you release your webapp, fields you've configured will suggest similar terms as users are typing in the QueryBuilder or the template form.

Optional constraints

To make a template constraint optional:

1. edit the template in the query builder
2. click on the padlock next to the constraint
3. select optional:

Required - the user must supply a value
Optional: ON - optional and ON by default
Optional: OFF - optional and OFF by defaul

Templates page

To have templates appear on the templates page, create a template as a SuperUser and tag the template with the "im:public" tag.

The templates are sorted by most popular first. If the user is logged in the user's most popular templates are shown first.

1.6.6 Query Results

Query results can be configured in a number of ways, including:

export

See `export` for details on exporting options.

column headers

See *Using Class and Field Labels* to change column headers.

links

Only unique fields (class keys) are links in results pages. Add to *Class keys* to make the fields links on results pages. Instead of linking to an intermine report page, you can set the links to redirect to external page. See `redirects`

weird brackets

You may see the following in query results: *GO:0007480 [GOTerm]*. This happens when a column is a parent type but the individual result is a subclass. The subclass will by in brackets.

The initial Page Size

This can be configured on a table by table basis when the table is initialised:

```
$('#my-table').imWidget({
  type: 'table',
  url: 'www.flymine.org/query',
  query: {from: 'Gene', select: ['*'], where: {symbol: 'foo*'}},
  properties: { pageSize: 20 }
});
```

Icons

Two different icon style are supported, bootstrap *glyphicons* and *fontawesome*. These differ in the underlying technology they use, one using images (glyphicons) and the other SVG fonts (fontawesome). By using fonts fontawesome icons generally look a bit nicer, but they are not compatible with IE8. For this reason *glyphicons* are the default, and *fontawesome* must be selected explicitly:

```
intermine.setOptions({icons: 'fontawesome'}, '.Style');
```

To apply this setting in your current web-app, see *Setting Javascript Options*.

The initial state of Sub-Tables

Outer-Joined collections are rendered in subtables within a single cell. By default these are not immediately rendered, and just the number of rows are indicated. This means that even sections with very large sub-tables are rendered efficiently - in the worst case the sub-tables may contain thousands of rows, and so a table with even 10 main rows might present 10,000 sub-rows or more, which can significantly impact browser performance (an example of this would be a table that contained publications with an outer-joined selection of genes; genome publications can list every gene in an organism, and this scenario easily leads to very large sub-tables).

However, if you don't like the default behaviour and would prefer for the sub-tables to be open when the main table is rendered onto the page, this is simply altered, through the following configuration snippet:

```
intermine.setOptions({SubtableInitialState: 'open'})
```

If you would like to set this property on a table by table basis, then you must set the *SubtableInitialState* property to *open*, in the same manner as you would for pageSize.

```
$('#my-table').imWidget({
  type: 'table',
  url: 'www.flymine.org/query',
  query: {
    from: 'Gene',
    select: ['*', 'pathways.*'],
    where: {symbol: 'foo*'},
    joins: ['pathways']
  },
  properties: { SubtableInitialState: 'open' }
});
```

Cell Formatters

The cells in each table can be configured to display their information in custom manners. To do this a javascript function must be registered to handle certain types of cell, and configured to respond to certain paths.

Formatters are not enabled by default, as they may be unexpected, and in could cause unneccessary requests to the server. Fortunately they are easily enabled. There are four formatter included (but not enabled) by default:

- Location - formats a chromosome location as eg: "2L:123..456"
- Sequence - formats a DNA or Protein sequence in FASTA lines.
- Publication - formats a publication in a citable format with title, first author and year.
- Organism - formats an organism's name in italics, using the short-name format.

To enable these formatters register the formatted path (see below), eg:

```
intermine.scope('intermine.results.formatsets.genomic', {
  'Organism.name': true,
  'Organism.shortName': true
});
```

To enable all the default formatters, you can use the following snippet:

```
var keyPath, formatsets = intermine.results.formatsets.genomic;
for (keyPath in formatsets) {
  formatsets[keyPath] = true;
}
```

Such customisation javascript should be placed in a custom model-includes.js file.

The Formatting Function

The interface expected for a formatting function is:

```
(Backbone.Model intermineObject) -> String|HtmlElement
```

Where the Model instance represents an intermine object. Fields of the object can be retrieved through the standard `#get(String)` method. The return value will be inserted into the table using the `jQuery#html` function, so both html strings and HtmlElements can be accepted as return values.

This function is executed as a method on a intermine.results.table.Cell (which will be bound as `this`), supplying the following properties as part of its interface:

```
this.el :: HtmlElement - The cell element in the DOM.
this.$el :: jQuery - The cached jQuery selector for the cell element.
this.options :: Object - The arguments supplied when constructing the cell, this includes:
  options.query :: intermine.Query
```

The function may also support two optional parts of the formatter interface:

```
Formatter.replaces :: Array<String> - The list of fields of the class that this formatter replaces.
Formatter.merge :: (Backbone.Model, Backbone.Model) -> () - A function to merge information
  from different objects into a single model.
```

A typical pattern would be to check to see whether the object currently has all the information required to render it, and if not then make a request to retrieve the missing information. Any changes to the model will cause the cell to be re-rendered, thus a request that gets missing information and sets it onto the model will cause the function to be called again with the complete information.

For examples of implementations of this interface please see:

- https://github.com/intermine/im-tables/blob/dev/src/formatters/bio/core/organism.coffee
- https://github.com/intermine/im-tables/blob/dev/src/formatters/bio/core/chromosome-location.coffee

The Formatting Configuration

To register a function to respond to specific types of data, it must be referenced under the `intermine.results.formatters` namespace by the name of the class that it handles. For example this can be done with the `intermine.scope` function:

eg:

```
intermine.scope('intermine.results.formatters', {Exon: myExonFormatter});
```

A separate entry must be made under the 'intermine.results.formatsets.{modelname}' namespace to register which paths trigger cell formatting. For example to register a formatter for the 'Exon' class which only formats the 'symbol' field:

```
intermine.scope('intermine.results.formatsets.genomic', {'Exon.symbol': true});
```

In a similar way, we can disable any currently configured formatter by setting the value of this value to 'false':

```
intermine.scope('intermine.results.formatsets.genomic', {'Exon.symbol': false});
```

individual formatters can be configured to respond to different fields of an object. So you could have one formatter for *Gene.length* and another for *Gene.symbol*, if you are unable to achieve what you need with css alone. To do this, the value in the formatset should be the formatter itself, rather than a boolean value, eg:

```
intermine.scope('intermine.results.formatsets.genomic', {
  'Gene.symbol': geneSymbolFormatter,
  'Gene.length': geneLengthFormatter
});
```

Branding

Links to your site (or others) can be branded with icons. This is configurable by setting option as follows:

```
intermine.scope('intermine.options.ExternalLinkIcons',
  {"http://myhostname": "http://myhostname/my-branding.png"}
);
```

All links in table cells with the prefix *http://myhostname* will use the given image as a logo.

This requires that *intermine.options.IndicateOffHostLinks* is set to true.

1.6.7 QueryBuilder

Select a Data Type to Begin a Query

types in bold Tag types with *im:preferredBagType* tag. Use the model browser to tag classes, eg. http://www.flymine.org/query/tree.do

intro text Most text in InterMine can be set in model.properties, see *Text and messages*.

help text Set in *classDecriptions.properties* file

query builder

SUMMARY Which columns appear when you click on SUMMARY button are set in WebConfigModel.

autocomplete Add fields to the `/database/database-building/post-processing/objectstore-summary-prope` file to have their form fields autocomplete.

Hiding fields

In your `webconfig-model.xml`, set a property `showInQB` for a `<fieldconfig />` to `true` to hide a field from a Class.

An example of hiding an attribute field:

```
<class className="org.intermine.model.testmodel.Manager">
    <fields>
        <fieldconfig fieldExpr="age" showInQB="false"/>
    </fields>
</class>
```

An example of hiding a Reference or a Collection field:

```
<class className="org.intermine.model.testmodel.Manager">
    <fields>
        <fieldconfig fieldExpr="address" showInQB="false"/>
    </fields>
</class>
```

1.6.8 Keyword Search

InterMine uses Solr for its keyword search index.

By default the index will include the text fields of all objects in the database. Each object in the database becomes a document in the index with text attributes attached. You can configure classes to ignore, such as locations and scores that don't provide text information. You can also add related information to an object, for example you can configure that the synonyms, pathways and GO terms should be included in the Gene's entry.

fields in the results determined by WebConfigModel

type class of object

score determined by the Lucene search, from 0 to 1

lists Users can make lists from search results but only if all results are of the same type.

To inspect the index directly: http://localhost:8983/solr/

Config file

The config file is located at *MINE_NAME/dbmodel/resources/keyword_search.properties*

- index.temp.directory
  - directory for search index
- index.references.<CLASS_NAME>
  - eg. index.references.Gene
  - index these objects' references in addition to the normal indexing
  - eg. if Gene.pathways is indexed so that when users search for pathways, the associated genes are also returned as search results
- index.ignore
  - do not index these classes
- index.ignore.fields
  - do not index these fields
  - eg *index.ignore.fields = SNP.type SNP.alleles*
- facets
  - Will appear as filters on the left panel in the search results
  - choose *single* for references, *multi* for collections
  - Note: you must index any references used as facets. (see: above at ''‘index.references’'').
- index.boost.<CLASS_NAME>
  - weight this class heavier than other objects
- search.debug
  - debug setting off, used only for testing
- index.optimize
  - boolean, defaults to false.
  - If set to *true*, reorganises the index so chunks are placed together in storage which might improve the search time. (Similar to defragmentation of a hard disk.) Requires an empty space in the storage as large as the index, and takes additional time.

Search Results

The fields displayed in the keyword search results are determined by the WebConfigModel file.

- If the fields are ClassKeys:
  - links in blue
  - shown at the top
- If the fields are not ClassKeys:
  - NOT linked, black text
  - shown below the links

Search Index

You can rebuild the search index by running this command in in your mine:

```
~/git/flymine $ ./gradlew postprocess -Pprocess=create-search-index
```

You need to re-release your webapp.

To inspect the index directly: http://localhost:8983/solr/

Solr

See *Solr* for details on how to install Solr.

Solr Partial String Match Configuration

In its default configuration, Solr will not match partial search terms. For example a gene named *REVOLUTA* will be returned in the search results for search term "REVOLUTA" but not for search term "REV." In order to have Solr return partial string matches, you must edit its configuration on the Solr server:

1. ADD the following to /var/solr/data/[mine]-search/conf/managed-schema. (This example implements it for hits against Gene.primaryIdentifier and Gene.secondaryIdentifier.)

```
<fieldType name="text_ngram" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.NGramFilterFactory" minGramSize="1" maxGramSize="50"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
<field name="gene_primaryidentifier" type="text_ngram" indexed="true" stored="true"/>
<field name="gene_secondaryidentifier" type="text_ngram" indexed="true" stored="true"/>
```

2. REMOVE the gene_primaryidentifier and gene_secondaryidentifier field definitions from the earlier part of the file. They look like this:

```
<field name="gene_primaryidentifier" type="analyzed_string" multiValued="true" indexed="true" require
<field name="gene_secondaryidentifier" type="analyzed_string" multiValued="true" indexed="true" requi
```

OR, simply UPDATE the existing records, replacing the parameters with: type="text_ngram" indexed="true" stored="true".

3. RESTART Solr to load the new config, e.g. under System V:

```
$ systemctl restart solr
```

4. REBUILD the search index using the Solr-related postprocesses:

```
./gradlew postprocess -Pprocess=create-search-index
```

Your keyword search will now return results on partial matches for the attributes that you configured in Solr (Gene.primaryIdentifier and Gene.secondaryIdentifier in this example).

1.6.9 General Layout

This page describes how to customise the look & feel of the whole InterMine webapp.

Parts

Logo

The logo is independent from any themes and is located here `MINE_NAME/webapp/src/main/webapp/model/images/logo.` The recommended size is 45px x 43px.

Menu Tabs

The tabs are set in InterMine's internationalisation file: `intermine/webapp/main/resources/webapp/WEB-INF/classes`

Each page has a name and a tab, for example:

```
mymine.tab = mymine
```

In addition to InterMine's file, each mine has its own internationalisation file: `MINE_NAME/webapp/src/main/resources/model.properties`. Properties set in this file overwrite the ones set in InterMine's `InterMineWebApp.properties`. Below is an example of how to add tabs to your mine. Replace "api" with the name of your new tab.

First, copy *headMenu.jsp* from InterMine to your local mine: `MINE_NAME/webapp/src/main/webapp`. Add your new tab.

```
<li id="api"  <c:if test="${tab == 'api'}">class="activelink"</c:if>>
  <a href="/${WEB_PROPERTIES['webapp.path']}/api.do">
    <fmt:message key="menu.api"/>
  </a>
</li>
```

Then add the text for that tab to your *MINE_NAME/webapp/src/main/resources/model.properties* file:

```
# HEADER
menu.api = API
```

You'll need to configure our web framework (Struts) to properly load your JSP page:

```
# in MINE_NAME/webapp/src/main/resources/struts-config-model.xml
<action path="/api" forward="api.page"/>

# in MINE_NAME/webapp/resources/tiles-defs-model.xml
<definition name="api.page" extends="layout.template">
    <put name="body" value="api.tile"/>
    <put name="pageName" value="api"/>
</definition>

<definition name="api.tile" path="/api.jsp"/>
```

Finally, add your JSP file to the *MINE_NAME/webapp/src/main/webapp* directory and re-release your webapp.

Keyword search box

This search box queries the search index created in the postprocess `create-search-index`. To change which placeholder identifiers will appear in the box, edit the *quickSearch.identifiers* property in *Database and Web application*.

See also:

Keyword Search for details on how to configure the search index.

Footer

`feedback.destination` in *Database and Web application* changes the recipient email address for contact form

`funding` in *Text and messages* changes the "funded by" text

`project.citation` in *Features* changes the "cite" text

Favicon

Favicon (icon seen next to the url of the webapp in the browser url bar) can be set by adding the following line:

```
<link rel="shortcut icon" type="image/x-icon" href="model/images/favicon.ico">
```

Into the `webapp/resources/webapp/layout.jsp` file and its `</head>` section. The favicon itself should be located in `<your_mine>/webapp/src/main/webapp/model/images/favicon.ico`.

If you want to generate a favicon from an image, use this Dynamic Drive tool.

Other properties

`project.sitePrefix` in *Database and Web application* configures the link

`project.title` in *Database and Web application* configures the name of the mine

`project.releaseVersion` in *Database and Web application* configures the version of the mine

`project.subTitle` in *Database and Web application* configures the subtitle showing in the header

`header.links` in *Features* configures the links in upper right corner

Changing look and feel, the theme

InterMine provides a set of default themes but you can also create your own. All themes are defined in /themes directory in InterMine. Explore the folder to see the themes available.

To switch a theme edit *Features*:

```
# web.properties
theme = purple
```

You need to change this property to the name of the theme you want to use (the directory name), then re-release the webapp. Be sure to run `./gradlew clean` first to ensure that all of the old files are deleted.

Developing your own theme

With CSS knowledge and open source image software such as Gimp or Inkscape you can develop your own theme. Each theme directory contains a `theme.css` file, which is broken down in annotated sections, and image files. The image files are required for displaying menus, headers and backgrounds and can be modified with image software to match your colour scheme. Create a new directory under `webapp/src/main/webapp/themes`, copy the contents of another theme directory into it and start editing.

1.6.10 Region Search

Configuration

struts-config-model.xml

```
<action path="/initGenomicRegionSearchOptions" type="org.intermine.bio.web.struts.GenomicRegionSearch
<action path="/genomicRegionSearch" forward="genomicRegionSearchOptions.page"/>
<action path="/genomicRegionSearchResults" forward="genomicRegionSearchResults.page"/>
<action input="/genomicRegionSearchOptionsBase.jsp" path="/genomicRegionSearchAction" name="genomicRe
  <forward name="genomicRegionSearchResults" path="/genomicRegionSearchResults.do" redirect="false"/>
  <forward name="genomicRegionSearchOptions" path="/genomicRegionSearch.do" redirect="true"/>
</action>
<action path="/genomicRegionSearchAjax" type="org.intermine.bio.web.struts.GenomicRegionSearchAjaxAct
```

tiles-defs-model.xml

```
<definition name="genomicRegionSearchOptions.page" extends="layout.template">
  <put name="body" value="genomicRegionSearchOptions.tile" />
  <put name="pageName" value="genomicRegionSearch" />
</definition>
<definition name="genomicRegionSearchOptions.tile" path="/model/genomicRegionSearchOptionsBase.jsp"
<definition name="genomicRegionSearchResults.page" extends="layout.template">
  <put name="body" value="/model/genomicRegionSearchResultsBase.jsp" />
  <put name="pageName" value="genomicRegionSearchResults" />
</definition>
```

struts-config-form-model.xml

```
<form-bean name="genomicRegionSearchForm" type="org.intermine.bio.web.struts.GenomicRegionSearchForm"
  <form-property name="organism" type="java.lang.String"/>
  <form-property name="featureTypes" type="java.lang.String[]"/>
  <form-property name="pasteInput" type="java.lang.String"/>
  <form-property name="fileInput" type="org.apache.struts.upload.FormFile"/>
  <form-property name="whichInput" type="java.lang.String"/>
  <form-property name="dataFormat" type="java.lang.String" initial="isNotInterBaseCoordinate"/>
  <form-property name="extendedRegionSize" type="java.lang.String"/>
</form-bean>
```

model.properties

```
genomicRegionSearch.title = Overlap features search from a new list of Genomic Regions
genomicRegionSearch.isNotMultipart = The request is not a file upload request
genomicRegionSearch.spanMisformatted = {0} is in a wrong format
genomicRegionSearch.spanFieldSelection = Please select some {0}
genomicRegionSearch.noSpanPaste = You need to type/paste in some genomic regions
genomicRegionSearch.noSpanFile = You need to type/paste in some genomic regions or select a file to u
genomicRegionSearch.isNotText = {0} is an invalid file type - file must be in plain text format
genomicRegionSearch.noSpanFileOrEmpty = The file you specified does not exist or is empty
genomicRegionSearch.spanInWrongformat = {0} is in a wrong format
genomicRegionSearch.spanInputType = Input type can't be solved
genomicRegionSearch.allRegionInvalid = All regions are invalid. Please do a new search.
genomicRegionSearch.organismEmpty = Organism is empty, please check the data is loaded.

genomicRegionSearchOptions.tab = genomicRegionSearch
genomicRegionSearchOptions.title = Genomic Regions Search
genomicRegionSearchResults.tab = genomicRegionSearch
genomicRegionSearchResults.title = Genomic Regions Search Results

menu.genomicRegionSearch = Regions
menu.genomicRegionSearchOptions = Genomic Region Search
menu.genomicRegionSearchResults = Genomic Region Search Results
```

web.properties

```
genomicRegionSearch.display = true
genomicRegionSearch.service =
genomicRegionSearch.optionsJavascript =
genomicRegionSearch.resultsJavascript =
genomicRegionSearch.optionsCss =
genomicRegionSearch.resultsCss =
## Make sure pre-defined organisms have chromosome location information in the database
genomicRegionSearch.defaultOrganisms = D. melanogaster
## Exclude feature types for all organisms, comma separated
genomicRegionSearch.featureTypesExcluded.global = GeneFlankingRegion,YouNameItClass
## Exclude feature types for specified organism, semi-colon separated
genomicRegionSearch.featureTypesExcluded.byOrganism = D. melanogaster:GeneFlankingRegion,YouNameItCla
genomicRegionSearch.defaultSpans = 2L:14615455..14619002\\n2R:5866646..5868384\\n3R:2578486..2580016
genomicRegionSearch.caption = Search for features that overlap a list of genome coordinates you ente
genomicRegionSearch.howTo = <ul>\
                              <li>Genome regions in the following formats are accepted:\
                                  <ul>\
                                      <li><b>chromosome:start..end</b>, e.g. <i>2L:11334..12296</i><
                                      <li><b>chromosome:start-end</b>, e.g. <i>2R:5866746-5868284</i
                                      <li><b>tab delimited</b></li>\
                                  </ul>\
                              <li>Both <b>base coordinate</b> (e.g. BLAST, GFF/GFF3) and <b>interbase
                              <li>Each genome region needs to take a <b>new line</b>.</li>\
                          </ul>
## Query fields when export results as csv/tsv
genomicRegionSearch.query.Gene.views = {0}.primaryIdentifier,{0}.symbol,{0}.chromosomeLocation.locate
genomicRegionSearch.query.Gene.sortOrder = {0}.chromosomeLocation.start asc
## 10,000 is the default value, only set if you want a different value
genomicRegionSearch.initBatchSize = 10000
```

- Update defaultOrganisms property as needed
- to disable genomic region search, set *genomicRegionSearch.display = false*
- also add *genomicRegionSearch* to *layout.fixed*, e.g.

```
layout.fixed = begin,template,templates,bag,customQuery,query,error,api,genomicRegionSearch
```

- add to ''genomic_precompute.properties'', note: do not duplicate the query number

```
precompute.query.30 = SELECT a3_.shortName AS a1_, a4_.class AS a2_ FROM org.intermine.model.bio.Orga

precompute.query.31 = SELECT a4_.class AS a1_, a5_.name AS a2_, a5_.description AS a3_ FROM org.inte
```

Region Search V2

Search page This page can be kept as it is, but the query can be constructed and sent to the server side by webservice. The Structs elements can be removed.

GenomicRegionSearchService This class has the methods to:

- generate data (JSON) for search page
- parse search form and valid input
- generate search queries (one region with one query)
- generate results table and download/galaxy links

This class can be basically replaced by webservices + html

Update IQL query to pathquery

Currently, region query is constructed by lQL (Intermine Query Language) due to lack of implementation on range constraint in pathquery at the time we developed it. Update IQL to pathqueries and send by webserive, the output will be a list of results tables or a single results table grouped by region.

See *GenomicRegionSearchUtil.java#L270-497*

Query fields:

In the IQL

See *GenomicRegionSearchUtil.java#L318-323*

In ResultRow

See *GenomicRegionSearchQueryRunner.java#L186-212*

In Results table

See *GenomicRegionSearchService.java#L1106-1112*

Polling

We create a synchronizedMap to hold all the query results and put it in an http request. On the results page, there is a checker (javascript) checking the size of the map, so a progress bar will be updating. The results table will be generated once 10 results return, the pager will be updated dynamically. he whole part will be replaced by InterMine results table.

See GenomicRegionSearchQueryRunner.java#L129-223

Results table and download links. Replaced by InterMine results table.

Adding the strand specific search option

Since InterMine 1.7, there is an additional feature on the Region Search page to restrict searches to a specific strand. The user activiates this by ticking a checkbox. For example, Chr1:12345-23456 indicates a region on the + strand; Chr1:23456-12345 indicates a region on the - strand. One situation in which this is useful is when you have a series of BLAST-generated regions on which you'd like to search for upstream gene flanking regions. In this case, there is no point in matching with gene flanking regions on the opposite strand.

However, adding this feature to the Region Search page requires a new entry in an InterMine installation's *struts-config-form-model.xml* file. A new InterMine installation will have this entry but existing updated InterMine installations will not. Therefore, to add this feature for an existing InterMine installation, the steps are to

1. Add a strandSpecific form property to the installations Region Search form in $MINE/webapp/resources/struts-config-form-model.xml

```
<form-bean name="genomicRegionSearchForm" type="org.intermine.bio.web.struts.GenomicRegionSearchForm"
    ...
    <form-property name="strandSpecific" type="java.lang.Boolean"/>
</form-bean>
```

2. Activate this by setting the following property in *web.properties*

```
genomicRegionSearch.enableStrandSpecificSearch = true
```

If this feature is not present or the checkbox is unchecked, then the default behaviour remains to search both strands.

1.6.11 Customise Web Application

Content

Database and Web application

InterMine is governed by a properties file located in the $HOME/.intermine named *$MINE_NAME.properties*. This page describes which values are set in that file.

Example: https://github.com/intermine/biotestmine/blob/master/data/biotestmine.properties

Database names and locations

The following properties determine the settings for the production database. This database is used by the build system and the webapp.

Property name	Example	Determines
db.production.datasource.serverName	server_name	server name
db.production.datasource.databaseName	biotestmine	database name
db.production.datasource.user	postgres_user	database username
db.production.datasource.password	SECRET	database password

The following properties determine the settings for the items database. This database is used during builds only.

Property name	Example	Determines
db.common-tgt-items.datasource.serverName	server_name	server name
db.common-tgt-items.datasource.databaseName	biotestmine	database name
db.common-tgt-items.datasource.user	postgres_user	database username
db.common-tgt-items.datasource.password	SECRET	database password

The following properties determine the settings for the user profile database. This database is used by the webapp only. It holds all user related information, including lists, queries and tags.

Property name	Example	Determines
db.userprofile-production.datasource.serverName	server_name	server name
db.userprofile-production.datasource.databaseName	biotestmine	database name
db.userprofile-production.datasource.user	postgres_user	database username
db.userprofile-production.datasource.password	SECRET	database password

Web application name and location

Property name	Example	Determines
os.production.verboseQueryLog	true	if true, all queries are logged. Defaults to false
webapp.deploy.url	http://localhost:8080	location of tomcat server
webapp.hostname	localhost	name of host
webapp.path	biotestmine	location of path of webapp
webapp.manager	TOMCAT_USER	tomcat username, needed to deploy webapp
webapp.password	TOMCAT_PWD	tomcat password, needed to deploy webapp
webapp.baseurl	http://www.flymine.org	home link; used by client side JavaScript AJAX requests
superuser.account	test_user@mail_account	account name for superuser
superuser.initialPassword	secret	password used when account is created
project.standalone	true	run with associated web site. Defaults to false
project.title	biotestmine	name of mine
project.subTitle	An example of InterMine.bio with data from <i>Plasmodium falciparum</i>	text that appears in the header at the top of the page
project.releaseVersion	tutorial	text that appears at the top of the page next to the mine name
project.sitePrefix	http://www.flymine.org	various URLs use this as the prefix
project.helpLocation	http://www.flymine.org/help	various URLs use this as the prefix

Warning: *webapp.baseurl* and *webapp.path* must be correct or else your queries will not run

Email

Emails are sent to users when they create an account, forget their password, or use the contact form.

Property name	Example	Determines
mail.host	localhost	mail host to use
mail.from	account@my_mail_host	"from" email address
mail.subject	Welcome to biotestmine	"subject" for email send when account created
mail.text	You have successfully created an account on BioTestMine	"body" for email send when account created
feedback.destination	test_user@mail_address	recipient of feedback form located on bottom of every page

This is the normal mailer. There is a different configuration for SMTP.

Multiple versions of a mine

It's possible to use several properties files by adding a suffix. Here's an example scenario:

1. add a suffix to the name of your property file:
   - *biotestmine.properties.dev* - points to the development database and a webapp
2. use *-Dorg.gradle.project.release=dev*

# *dev* is the suffix on the properties filename

```
# build the database specified in dev properties file
    ./gradlew builddb -Dorg.gradle.project.release=dev

# deploy the webapp specified in dev properties file
    ./gradlew cargoReDeployRemote -Dorg.gradle.project.release=dev
```

Text and messages

These files control much of the text in the web application:

InterMineWebApp.properties Most of the text appearing on the webapp (button names, forms, some help text, etc.) is defined in this file. If you want the webapp to appear in a different language than English, you will have to translate the file. This file is located in the InterMine webapp JAR.

model.properties Model specific properties. Merges with InterMineWebApp.properties, overwrites properties in that file.

Features

The *web.properties* file configures several attributes for the InterMine web application.

attributeLink Used to configure hyperlinks, generally to external dbs. See "External Links" section below

bag.example.identifiers Text present in the list upload form. See "List upload examples" section below

externallink Redirect links in query results. See `/webapp/query-results/redirects`

galaxy See *Galaxy*

genomicRegionSearch See *Region Search*

header.links links at upper right corner

meta.keywords will populate meta tag for keywords

meta.description will populate meta tag for description. Google uses this in their search results, I think

project.citation populates the "Cite" text in the footer.

portal.welcome the message to show when a user arrives at the webapp via the portal action (eg. <something>/portal.do)

quickSearch.identifiers Text displayed in search box

theme Colour scheme for the webapp. Available options are: blue, bright_blue, gold, green, grey, brown, ecoli_blue, metabolic, modmine, ratmine and purple

xrefLink Used to configure hyperlinks for CrossReferences. See below

markup.webpages.enable Used to enable structured data in JSON-LD format in InterMine web pages. Available options are: true or false

Branding

These parameters are returned by the branding API end point, and are used by external applications, e.g. the InterMine iOS app, the InterMine registry and the InterMine R client.

branding.images.logo	This image should be 45px by 45px
branding.colors.header.main	Main colour for your mine, defaults to grey, #595455
branding.colors.header.text	Text colour for your mine, defaults to white, #fff

Home page

Search box (first box on the left)

begin.searchBox.title	title of box on left
begin.searchBox.description	text in the box on the left
begin.searchBox.example	text in the form field

List upload box (middle box)

begin.listBox.title	Title of box
begin.listBox.description	Text in box
bag.example.identifiers	Text in form field

Third box

begin.thirdBox.title	Title of box if user is new
begin.thirdBox.visitedTitle	Title of box if user has visited before
begin.thirdBox.description	Text in box
begin.thirdBox.linkTitle	Text for large button
begin.thirdBox.link	URL for large button

Tabs

Templates tagged with each category will appear under the appropriate tab.

begin.tabs.1.id	Name of category, eg. Genes
begin.tabs.1.description	Text for that tab

List upload examples

Using the *bag.example.identifiers* key, one can provide a list of keyword examples on the list create/upload page. This could lead to a mixed list of items being updated and only, say Protein or Gene, identifiers being uploaded.

External links

You can add links to other websites by adding entries to the *web.properties* file.

The format for this property is:

```
# on the report page - a single identifier
'attributelink' + unique_name + class + taxonId + attributeName + (url|imageName|text)

# on the list analysis page - a list of identifiers
'attributelink' + unique_name + class + taxonId + attributeName + 'list' + (url|imageName|text)
```

unique_name used to distinguish between multiple configurations for the same attribute/organism/class combination

class class of object to link, eg. Protein

taxonId either a proper id or '*' when no assumptions is made regarding the organism

attributeName which identifier field to pass to the URL, e.g. if attributeName is primaryIdentifier, the value of primary identifier field will be used as the attribute value

list indicates the link will have a list of identifiers

url url to link to

imageName name of logo (optional), must be in /model directory

text text that will appear next to the logo

The value of the attribute (for the current object) is substituted anywhere the string "<<attributeValue>>" occurs in the text or the url

example:

```
attributelink.flybase.Gene.7227.primaryIdentifier.url=http://www.flybase.org/.bin/fbidq.html?<<attrib
attributelink.flybase.Gene.7227.primaryIdentifier.text=FlyBase: <<attributeValue>>
```

In this case *Gene* pages for Drosophila melanogaster will have a link that uses the *organismDbId* field.

A list example:

```
attributelink.flymine.Gene.*.primaryIdentifier.list.url=http://www.flymine.org/flymine/portal.do?exte
attributelink.flymine.Gene.*.primaryIdentifier.list.text=FlyMine
attributelink.flymine.Gene.*.primaryIdentifier.list.imageName=flymine_logo_link.gif
attributelink.flymine.Gene.*.primaryIdentifier.list.usePost=true
```

Only if a taxonId is specified the code will check if the link to the external db is relevant.

Settings for the xrefLink property

You can configure the URLs for querying CrossReference from external sources by adding entries to the *web.properties* file.

The format for this property is:

```
# on the report page
'xreflink' + dataSource_name + (url|imageName)
```

dataSource_name the name of the external database

url url to link to

imageName name of logo (optional), must be in /model directory

example:

```
xreflink.PFAM.url=http://pfam.sanger.ac.uk/family?
xreflink.PIRSF.url=http://pir.georgetown.edu/cgi-bin/ipcSF?id=
```

Cross references represent identifiers used in external databases, eg. FlyBase, UniProt. An object in InterMine which has CrossReference will have a identifier and data source for that cross reference. In order to find the cross reference in that data source, a url is required to link to and the full path should look like url+identifier, e.g. ''http://pfam.sanger.ac.uk/family?PF00001''. In web.properties, the first part of the full path could be configured as in "url", and identifier will be added programmatically to the rear of it. The dataSource_name should be consistent with the source name of the CrossReferences in the InterMine database.

OpenAuth2 Settings (aka. OpenID Connect)

You can configure your mine to accept delegated authentication from one or more identity resources which are protected by OAuth2 authentication. This involves sending the user to another site, having them sign in there, and being sent back to your InterMine with their credentials.

We are using the Apache OLTU library to help manage the authentication flow. This means that configuring some of the more common providers, such as Google, Facebook, Github and Microsoft is very simple. It also allows us to add any identity provider that meets certain minimum sanity requirements.

Warning: Google has closed down their OpenID-2 based authentication solution in favour of OpenID Connect (OAuth2). If you want to use Google as an authentication provider you must use OAuth2.

Configuration is managed through adding values to the `web-properties`.

Registering your Application. You *must register your application* with the provider, giving them details of your application such as its name, and where it will be located. This varies from provider to provider - see this tutorial for a good guide to the registration process for a number of popular providers. For example, for Google, you will need a Google+ account and to visit the Google developer's console to create an application.

For ELIXIR, you will need:

1. an ELIXIR identity. Please register the ELIXIR ID here , if you don't already have it
2. register the new client here, using the *Self-service client registration* page.
3. send an email to aai-contact@elixir-europe.org in order to receive a form that you have to completed with additional informations

For each application you will need to register the callback URI, which looks like:

```
${webapp.baseurl}/${webapp.path}/oauth2callback.do?provider=${PROVIDER}
```

Where *webapp.baseurl* and *webapp.path* are the corresponding values from your configuration, and *PROVIDER* is the name of the provider in all uppercase letters (as configured below). Google requires the *provider* parameter as part of the URI, but other providers do not - you should check with each of them.

You will probably be asked to register a javascript domain. This is not used by us, but you can enter the *webapp.baseurl*.

Enabling Supported Providers You will need to inform the InterMine system of the names of the providers which have been configured to work with your application. This should be a comma separated list of provider names. The values are case insensitive, and will be processed as upper-case values. E.G.:

```
# in  ~/.intermine/MINE.properties
# You can list just a single provider:
oauth2.providers = GOOGLE
# or multiple providers, combining standard and custom providers:
oauth2.providers = GOOGLE,ELIXIR,GITHUB,FACEBOOK,MICROSOFT,STRAVA,AIP
```

Configuring OLTU Supported Providers To configure an OLTU supported provider (such as Github or Facebook), you simply need to define the client-id and client-secret you registered your application with, eg:

> **Warning:** All secrets, including these ones (especially the client-secret) MUST not be committed to version control or made publicly accessible. DO NOT add them to your web.properties file, but instead add them to your mine.properties file (eg. ~/.intermine/MINE.properties).

```
# ~/.intermine/MINE.properties
oauth2.GITHUB.client-id = $GH-CLIENT-ID
oauth2.GITHUB.client-secret = $GH-CLIENT-SECRET
```

Configuring a Custom Provider To configure a custom provider some other properties need to be provided. Taking AIP's araport system as an example, this can be configured thusly:

```
# All OAuth2 clients need this configution. Do not commit to version control!
oauth2.AIP.client-id = YOUR_CLIENT_ID
oauth2.AIP.client-secret = YOUR_CLIENT_SECRET
```

The URLs needed by the flow - contact your provider to find these out:

```
oauth2.AIP.url.auth = https://api.araport.org/authorize
oauth2.AIP.url.token = https://api.araport.org/token
```

The scopes need to access the identity resource. This should include sufficient levels of permission to access the name and email of the authenticating user.

```
oauth2.AIP.scopes = PRODUCTION
```

Information about the way the token endpoint functions. If the token endpoint expects parameters to be passed in the query-string use the value "QUERY", and if the endpoint expects the parameters to be passed in the message body provide the value "BODY":

```
oauth2.AIP.messageformat = BODY
```

Information about the way the token endpoint responds. If the token endpoint responds with `JSON`, then provide the value "JSON", and if the endpoint responds with url-encoded form-data, then provide the value "FORM"

```
oauth2.AIP.responsetype = JSON
```

Information about the way the identity resource operates. If the resource expects the bearer token to be in the query parameters provide the value "query", and if the bearer token is expected to be in the `Authorization` header, pass the value "header".

```
oauth2.AIP.resource-auth-mechanism = header
```

The location of the identity resource. This must be a resource that can respond with `JSON`. If query parameters are needed they should be included in the URL. An `Accept` header will be provided with the value `application/json`.

```
oauth2.AIP.identity-resource = https://api.araport.org/profiles/v2/me
```

Guides to interpreting the response from the identity resource. These are all optional.

```
# Provide a value if the identity is within a message envelope. The value is the
# key of the envelope.
oauth2.AIP.identity-envelope = result
# Provide a key to access a unique identifier for the user. Default = id
oauth2.AIP.id-key = uid
```

```
# Provide a key to access the user's email. Default = email
oauth2.AIP.email-key = email
# Provide a key to access the user's name. May be a composite value (comma separated). Default = name
oauth2.AIP.name-key = first_name,last_name
```

Delegated Authentication with JWTs

InterMine supports completely automated delegated authentication, whereby a mediator may add a token that authenticates the user according to a chain of trust. This uses public-key cryptography to establish trust, and JWTs to transmit assertions.

Note: All the configuration in this section can (and should) go in your *~/.intermine/MINE.properties* file

To enable this feature you need to do a couple of things:

Create a Key Store [optional] InterMine needs access to public keys - this can mean creating a JKS key store (http://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html) with the certificate used to sign the JWTs - you should store the certificate against the alias with the same name as used in the *iss* claim in the JWT. The keystore file should be saved as *keystore.jks.$release* in the *~/.intermine* directory, or moved as part of your release cycle to *MINE/resources/webapp/WEB-INF/* immediately prior to building your webapp.

If you do this, then you need to provide the following configuration:

security.keystore.password	The password for this keystore.

If your keystore has no password, then you do not need to set that property. See below for a quick guide to creating a valid keystore.

Provide Public Keys in your properties files [optional] Instead of (or in addition to) creating a keystore, you can also provide keys in property files. Even though these are public keys, they are best included in your *~/.intermine/MINE.properties.release* file, since they will be specific to a particular instance. Internally if you do not provide a keystore, an empty one will be created.

This is done by listing them as follows:

security.publickey.$ALIAS	$BASE64_ENCODED_PUBLIC_KEY

You can provide multiple keys and they will be all stored in the applications key-store under the given alias. Every key must have an alias, even if there is only one. If there is a problem with the key (it cannot be decoded, it is not valid, etc) it will by default be skipped, unless the following property is set to *true* (in which case it will throw an error and prevent your web-application from starting):

keystore.strictpublickeydecoding	*true* or *false*

The value *BASE64_ENCODED_PUBLIC_KEY* is the base64 encoding of the bytes of public key. Below is a sample program to illustrate how to do this in Java and python:

```
import java.security.KeyPairGenerator;
import java.security.PublicKey;
import org.apache.commons.codec.binary.Base64;

public class EncodeKey {

    public static void main(String... args) throws Exception {
        PublicKey key = getKey();
        Base64 encoder = new Base64();
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        System.out.println(encoder.encodeToString(key.getEncoded()));
    }

    private static PublicKey getKey() {
        // Generating a random key - provide your own of course.
        return keyGen.generateKeyPair().getPublic();
    }
}
```

or

```
# using pycrypto https://www.dlitz.net/software/pycrypto/
from Crypto.PublicKey import RSA
from Crypto import Random

# Generate a new random public key.
random = Random.new().read
pair = RSA.generate(1024, random.read)
public_key = pair.publickey()

print(base64.encodestring(public_key.exportKey(format = 'DER')))
```

Selecting keys at runtime. Since this feature relies on public key cryptography, you need to tell the InterMine application which keys to use to verify which JWT tokens. This can be done with the following properties:

jwt.verification.strategy	*NAMED_ALIAS* (default), *ANY*, or *WHITELIST* - optional

This property defaults to the most secure option, *NAMED_ALIAS*, where only keys associated with the issuer of the token with be used to verify it. This means you will need to link the two. Each token must identify its issuer (with the *iss* claim), you can map from that value to a key available to InterMine by providing the alias it is available as in the keystore. If you plan on accepting your own tokens, then you can provide the alias of your private key.

security.keystore.alias.$iss	The alias for the key certificate used to sign the JWT.

If you use the *WHITELIST* strategy, the you must provide the list of aliases that can be used to verify JWTs. All of them will be tried until one verifies successfully.

jwt.alias.whitelist	The comma separated list of aliases to use.

If you select the *ANY* strategy, no further configuration is needed.

Multiple issuers can be supported by providing a key for each alias.

Managing non-standard claims InterMine reads to claims about the end user from the JWT - who it identifies, and their email address. The email claim is non-standard, and needs to be configured. The subject claim can be overriden if the JWT tokens you are receiving have their subject identified in a different claim. To do so provide the following properties (in the following table, *$iss* is the value of the *iss* claim of the token):

jwt.key.email.$iss	The name of the claim that provides the email of the subject. Defaults to *http://wso2.org/claims/emailaddress*
jwt.key.sub.$iss	The name of the claim that provides the identity of the subject. This should be unique for each issuer. Not needed if the token provides the *sub* claim

Other properties The following properties may also be important

jwt.publicidentity	Used as the *iss* claim on any tokens the application issues itself. Also, if the tokens received include an *aud* claim (see aud definition) then this value must match that value for verification to complete. Defaults to your project title.
jwt.verifyaudience	*true* or *false* (default = true). Whether to verify the *aud* claim.
security.privatekey.password	Used to gain access to the private key used by the application for signing its own tokens.
security.privatekey.alias	Used to retrieve the private key used by the application for signing its own tokens. To provide a private key you must configure a key store.

Checking your configuration An ant task is provided to make checking this (admittedly rather complex) set-up easier. To make use of it you should configure your keys as for production, acquire a valid JWT representative of one of the ones you expect to encounter in production, enter you webapp directory (*$MINE/webapp*) and then call the following ant task:

```
ant verify-jwt \
    -Drelease=$RELEASE \ # Needed to read the correct properties file
    -Dkeystore=$KEYSTORE_LOCATION \
    -Djwt=$JWT
```

If correctly set up, you should get a message printed to the console telling you who the token identifies.

Setting up the Key-Store You will need a Java Key Store to use public-key cryptography for security. To get started you can use the following command to generate a *keystore.jks* file with a new public/private key-pair:

```
keytool -genkey -alias ALIAS_A -keyalg RSA -keystore keystore.jks -keysize 2048
```

The following command will allow you to add a certificate to your key-store:

```
keytool -import -trustcacerts -alias ALIAS_B -file B.crt -keystore keystore.jks
```

This set-up would allow you to start accepting JWT tokens signed by the owner of *B.crt*, which could be configured by making sure they are associated in your property files. So if the owner of *B.crt* identified themselves with the *iss* (issuer) claim *http://b.com*, then you could link the certificate to the claim with the following property:

```
security.keystore.alias.http://b.com = ALIAS_B
```

Overriding properties

- *intermine/webapp/main/resources/webapp/WEB-INF/global.web.properties* - used by all mines. Properties set here will be available to everyone, even the test model mine.
- *bio/webapp/resources/webapp/WEB-INF/bio.web.properties* - used by all bio-mines. Properties set here will be available to all mines that use the bio layer. so not the test model model. Can overwrite properties in the global.web.properties file.
- *flymine/webapp/resources/web.properties* - used by a mine. Properties set here will be available to only that specific mine. Can create mine-specific properties or overwrite properties in the above two files.
- *$HOME/.intermine/flymine.properties* - used by a mine. Properties set here will be available only to that specific mine, and will override all other properties. Put sensitive values here that should not be commited to version control.

Data and Widget Configuration

The *webconfig-model.xml* file configures aspects of how data appears on the InterMine webapp.

This file allows for inheritance - a subclass will inherit from its parent class but only if that subclass has no configuration. Configuration settings for the parent class do not overwrite settings for the subclass.

Field Configuration

You can configure which fields are displayed on report and result pages for each class in your model.

attribute name	purpose	required?	default
fieldExpr	field name	yes	•
label	human readable name	no	generated automagically
showInInlineCollection	show field in inline collection (on report pages)	no	true
showInSummary	add field to query when user clicks on 'Summary' button in QueryBuilder	no	true
showInResults	show field in results table	no	true
outerInSummary	configure outer-joins when user clicks on 'Summary' in QueryBuilder	no	false
doNotTruncate	don't truncate display	no	false
fieldExporter	specify class to export file field	no	•
sectionOnRight	show on the right side of the page	no	false
sectionTitle	if sectionOnRight="true", title for section on right	no	•
openByDefault	if sectionOnRight="true", whether or not this section should be open	no	false

For example:

```
<class className="org.flymine.model.genomic.Protein">
  <fields>
    <fieldconfig fieldExpr="primaryIdentifier"/>
    <fieldconfig fieldExpr="primaryAccession"/>
    <fieldconfig fieldExpr="organism.name"/>
    <fieldconfig fieldExpr="length" displayer="/model/sequenceShortDisplayerWithField.jsp" />
  </fields>
  <bagdisplayers>
   < -- attribute links can now be displayed on protein list analysis pages -->
   <displayer src="attributeLinkDisplayer.tile"/>
  </bagdisplayers>
</class>
```

Displaying Data on Report pages

ReportDisplayers allow custom display of particular data types on report pages, typically to replace default tables with more appropriate presentation of data.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.CytoscapeNetworkDisplayer"
                 jspName="model/cytoscapeNetworkDisplayer.jsp"
                 replacesFields="interactions"
                 placement="Interactions"
```

Export Configuration

Users can export data from InterMine in comma or tab-delimited files. InterMine also allows for the addition of custom exporters. To add a custom exporter, create a Java class to format the data and add an entry to the web config file, for example:

```
<tableExportConfig id="sequenceExporter" actionPath="/exportAction?type=sequenceExporter"
                   className="org.intermine.bio.web.export.SequenceHttpExporter"/>
<tableExportConfig id="gff3Exporter" actionPath="/exportAction?type=gff3Exporter"
                   className="org.intermine.bio.web.export.GFF3HttpExporter"/>
```

Widget Configuration

At the bottom of the config file are the configuration entries for widgets. Please see [wiki:Widgets] for detailed information about how to configure widgets.

```
<enrichmentwidgetdisplayer id="publication_enrichment"
                           title="Publication Enrichment"
                           description="Publications enriched for genes in this list."
                           label="Publication"
                           startClass="Gene"
                           startClassDisplay="primaryIdentifier"
                           enrich="publications.title"
                           enrichIdentifier="publications.pubMedId"
                           constraints="organism.name=[list],primaryIdentifier  = null"
                           typeClass="org.intermine.model.bio.Gene"
                           views="secondaryIdentifier, symbol, organism.name,
                                 publications.title, publications.firstAuthor,
                                 publications.journal, publications.year, publications.pubMedId"
                           externalLink="http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&a
```

Class keys

Specify keys for the classes in your data model by adding them to the *class_keys.properties* file. This lets the webapp know how to uniquely identify objects of these classes. Users can then *upload* lists of such objects.

Keys defined in the *class_keys.properties* file are also used to boost the search visibility of their associated classes.

The *class_keys.properties* file specifies the keys used to generate the permanent navigable URL which is used in the "SHARE" button in the report pages. If not specified, the primaryidentifier key is used.

key	value
Pathway_URI	identifier

Given the above configuration, in FlyMine, the URL of the report page for the pentose phosphate pathway with identifier 00030, will be http://flymine.org/flymine/pathway:00030. No need to specify the keys for the core model classes (e.g. protein, publication...).

See *Permanent URLs* for details on permanent URLs.

See FlyMine's class keys for an example class keys file.

Setting Javascript Options

Many of the javascript tools included in InterMine pages can be customized at run-time with specific options. To do this the recommended practice is to include a custom set of option configurations in a javascript file that is included in your InterMine instance, or other embedding page. Do do this withing the context of the InterMine web-application, we recommend following the following steps:

- Create a new javascript file, named something like *model-includes.js*, and place it in the *MINE_NAME/webapp/src/main/resources* directory.
- Add your options to the file (see below).
- Configure your mine to load this file on every page (see below).

Adding options to the file

If for instance you wanted to configure the result-tables so that their cell-previews appeared on 'hover' rather than on 'click', which is the default, and also to enable the 'ChromosomeLocation' formatter, you would want the contents of your options file to be something like:

```
(jQuery(function() { // run when the page has loaded.
  if (intermine) {   // but only if there is something to do.
    intermine.setOptions({CellPreviewTrigger: 'hover'});
    intermine.setOptions({
      'Location.start': true,
      'Location.end': true
    }, 'intermine.results.formatsets.genomic');
  }
});
```

Configuring your mine to load your custom file

In one of your properties files (ideal your model web properties file), add a property beginning with *head.js.all.* that names this file. If your file is *my-mine-options.js*, then this line might look like:

```
head.js.all.MY_JS_OPTIONS = my-mine-options.js
```

1.6.12 Data Categories

Data category pages include various aspects of a concept on a single page. Category pages include:

- logo
- short description
- external links
- bulk download queries
- template queries
- direct links to QueryBuilder

All of the above should relate to a single concept, eg. Genomics or Interactions.

Data categories are defined in *aspects.xml*.

Aspects.xml

```xml
<aspect name="Genomics">
  <subtitle>Genome annotation</subtitle>
  <icon-image>model/genomics.gif</icon-image>
  <large-image>model/genomics.gif</large-image>
  <intro-text>
    The gene structure and other genome annotation in FlyMine are provided by
    a variety of curated source databases.  In most cases FlyMine includes
    all genome features that have chromosomal locations (eg. genes and repeat regions).
  </intro-text>
  <tile-name>model/genomics.jsp</tile-name>
  <aspect-source name="FlyBase" url="http://www.flybase.org"/>
  <aspect-source name="Ensembl" url="http://www.ensembl.org/Anopheles_gambiae"/>
</aspect>
```

Configuration

- logo
  - <icon-image>model/genomics.gif</icon-image> - appears on the home and data category pages
  - <large-image>model/genomics.gif</large-image> - appears on the individual data category page
- short description
  - <intro-text>TEXT HERE</intro-text>
  - appears on the top of the data category page
- external links
  - <aspect-source name="FlyBase" url="http://www.flybase.org"/>
  - appear on the top right corner of the data category page
- bulk download queries
  - appear on the top right corner of the data category page
- template queries
  - appear on the data category page
- direct links to QueryBuilder
  - links will appear at the bottom of the data categories page

To configure which template queries appear on a data category page, tag the template.

Data page/tab

The data tab points to this JSP file *intermine/webapp/main/resources/webapp/dataCategories.jsp*. You can overwrite this file and display your own customised file by putting a JSP in your */webapp* directory.

1.6.13 Web pages markup

We have applied structured data in JSON-LD format to InterMine web pages (using Bioschemas.org types and profiles), to improve findability so search engines can give more relevant results to users.

The markup are disabled by default, to enable them set the property *markup.webpages.enable* to *true* in the *web.properties file*.

We have applied the following markup:

Type	Applicable
DataCatalog	Main Home Page
DataSet	Report page for entitites with type DataSet

Home page markup

property	description	example
identifier	The identifier for the mine instance, based on the namespace assigned in the intermine registry [15]	https://registry.intermine.org/flymine
name	The name of the InterMine instance	FlyMine
descrition	The description of the InterMine instance	An integrated database for Drosophila and Anopheles genomics
url	The url of the InterMine instance	http://flymine.org
dataset	The list of the datasets stored in the InterMine instance containing name and url	

Report page markup for DataSet

property	description	example
name	The name of the dataset	FlyAtlas
description	The description of the dataset	Affymetrix microarray-based atlas
identifier	The url of the dataset, if provided, or the permanent URL of the report page	http://www.flyatlas.org/
url	The permanent URL of the report page	http://flymine.org/flymine/dataset:flyatlas

1.6.14 Help

This page lists how you can update the help sections of your InterMine.

Top Links

To add help links to the top of your website, add an entry to *web.properties* listing the links:

```
header.links=link1, link2
```

Then specify the URLs:

```
header.links.link1=http://www.mysite.com/link1
header.links.link2=http://www.mysite.com/link2
```

[15] When an InterMine instance is added to the registry, an unique and persistent namespace is assigned by the administrator. Some examples of namespaces: flymine, humanmine, flymine.beta. The identifier will be: https://registry.intermine.org/{namespace}. These identifiers are actionable, so if you put https://registry.intermine.org/{namespace} in the address bar of your browser, you will be redirected to URL set in the registry for the FlyMine. If the InterMine instance is not register, the url will be used instead.

For example, see FlyMine's web.properties file:

```
header.links=help,FAQ,about,cite,software
header.links.FAQ=http://trac.flymine.org/wiki/FlyMineFAQ
header.links.about=http://blog.flymine.org/?page_id=35
header.links.cite=http://blog.flymine.org/?page_id=37
header.links.help=http://blog.flymine.org/?page_id=45
header.links.software=http://blog.flymine.org/?page_id=39
```

Take a tour link

The tour link is set in *headMenu.jsp* as:

```
<project.helpLocation>/tour/start
```

Set *project.helpLocation* property in your mine.properties file. If you don't have help pages set up, link to FlyMine's pages:

```
project.helpLocation=http://www.flymine.org/help
```

Contextual help, the *?* on each page

Set the URL in your properties file

On each page is a ? that links to help pages. Specify the main URL that this question mark should link to by setting the *project.helpLocation* property in your mine.properties file.

If you don't have help pages set up, link to FlyMine's pages:

```
project.helpLocation=http://www.flymine.org/help
```

Set the context

1. If the user is on a webpage defined in the properties file, then when they click the help link they will be forwarded to the help section for the page they were viewing.
2. If the page they are on is not specified in the properties file, they will be forwarded to the first page of the help document.
3. The context is determined by parsing the URL and taking the name of the current webpage, minus the *.do*. For example, go to FlyMine and click on the 'templates' tab, this is the URL: http://www.flymine.org/query/templates.do. The parsed name of that webpage is "templates".
4. Below are the mappings from parsed webpage name to anchor names on the help page.

```
help.page.<parsed webpage name> = <anchor in help.html file>

help.page.begin=begin
help.page.templates=templates
help.page.bag=lists
help.page.bag.upload=lists:upload
help.page.bag.view=lists:view
help.page.customQuery=customQuery
help.page.mymine.lists=mymine:lists
help.page.mymine.history=mymine:queryHistory
help.page.mymine.saved=mymine:savedQueries
help.page.mymine.templates=mymine:savedTemplates
help.page.mymine.password=mymine:changePassword
help.page.dataCategories=data
help.page.objectDetails=reportPage
help.page.template=template
help.page.results=results
help.page.bagDetails=listAnalysis
help.page.bagUploadConfirm=buildList
help.page.query=query
help.page.importQueries=importQueries
help.page.importTemplates=importTemplates
help.page.tree=tree
help.page.aspect=dataCategory
```

Your mine's web.properties file is merged with this web.properties file, so entries you add to web.properties will overwrite the values listed above.

Data definitions

Update these in the classDescriptions.properties file.

1.6.15 Linking in to your mine

This page aims to describe the various ways to link to a Mine.

Link directly to query results

Template name

Links to results of specified template. URL generated on template form in webapp.

http://www.flymine.org/query/loadTemplate.do?name=Chromosome_Gene&constraint1=Gene.chromosome.primaryIdentifier&op1=eq

Make sure to include *&method=results* at the end of the query string.

Query XML

Links to results of query. Can run any query built by QueryBuilder; QueryBuilder generates the XML.

http://www.flymine.org/query/loadQuery.do?skipBuilder=true&query=%0A%3Cquery+name%3D%22%22+model%3D%22genomic%

Link to List Analysis page

Template results

Links to list analysis page comprised of results of template query. "path" attribute determines which column used to create list. URL available on template form in webapp.

http://www.flymine.org/query/loadTemplate.do?name=Pathway_Genes&constraint1=Pathway.name&op1=eq&value1=Pentose+phosph

List of Identifiers

Links to list analysis page for specified objects. For a very long list, use a form instead of a link. Can use any identifiers.

http://www.flymine.org/query/portal.do?externalids=CG2262,CG3069,CG2859,CG5041,FBgn0036513&class=Gene

Query builder

Links directly to query builder, starts a query using the provided list.

http://beta.flymine.org/beta/loadQuery.do?name=copy&method=list

Report page

Links directly to report page. URL available on report page in webapp.

http://www.humanmine.org/humanmine/portal.do?externalids=pparg&class=Gene&origin=readthedocs

Optionally, add *extraVlue* parameter with the organism name, e.g.:

http://www.humanmine.org/humanmine/portal.do?externalids=pparg&class=Gene&extraValue=H.+sapiens

Link into Mine with Orthologues

The example URL contains ‘’D. melanogaster” genes. The results will contain the corresponding ‘’C. elegans” genes, if any. This will only work if you have orthologue data loaded into your Mine. Will forward to report page OR list analysis page.

http://www.flymine.org/query/portal.do?externalids=CG2262,CG3069,CG2859,CG5041,FBgn0036513&class=Gene&orthologue=C.%

Convert any identifiers to Genes

When linking to a report page or a list analysis page you can convert the data type, for instance if you provide a Protein identifier and want to link to the corresponding Gene, you need to specify the class as Gene. Will only work if you have a converter template available

http://www.flymine.org/query/portal.do?externalid=EVE_DROME&class=Gene

More examples

See FlyMine for more examples: https://intermineorg.wordpress.com/flymine/link-to-flymine/

1.6.16 Third party tools

Cytoscape network viewer

This tool takes gene interaction data from Intermine and visualises it using cytoscape.js, a fabulous network visualisation tool. It replaces the flash-based CytoscapeWeb network viewer found in previous versions of the tool.

Configuration

1. add the following entry to your ‘‘‘webconfig-model.xml’‘‘ file:

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.CytoscapeNetworkDisplayer"
         jspName="model/cytoscapeNetworkDisplayer.jsp"
         replacesFields="interactions"
         placement="Interactions"
         types="Gene,Protein"/>
```

2. If you host your own Intermine CDN, make sure to pull the most recent update, as the interaction displayer script is loaded via CDN, under *js/intermine/gene-interaction-displayer*.
3. re-release your webapp and you should see the interaction displayer on gene report pages.

Data export format

The network visualisation can be exported as:

- PNG
- JPG
- TSV
- CSV

Implementation

This tool accesses the list of gene interactions for the target gene by calling a web service, sorting the data into edges and nodes, and inserting them into an HTML canvas for display. It can also be used externally to the report page as a stand alone application. For external setup instructions, see the Cytoscape Intermine repo, and the standalone app demo page

Dependencies: This tool uses imjs to query the data, and imtables to display table data.

A short list of Java files found on the Intermine side:

CytoscapeNetworkDisplayer.java the report displayer class, get a set of genes interacting with the report gene, in your case, the genes/proteins on the same pathway as the report gene/protein

CytoscapeNetworkDisplayer.jsp the web page to display the network

CytoscapeNetworkService.java service class

EsyN

A network viewer that you can place on your gene report and list pages.

Users can click on the links to follow the data to esyn.org, and construct interaction networks and models of biological processes using publically available data.

Configuration

Report page Add the following entry to your *webconfig-model.xml* file:

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.EsynDisplayer"
                 jspName="model/esynDisplayer.jsp"
                 replacesFields=""
                 placement="summary"
                 types="Gene"/>
```

List analysis

1. add the following entries to your *struts-config-model.xml* file:

```
<action path="/initEsynListDisplayer" type="org.intermine.bio.web.EsynListDisplayer"/>
```

2. add the following entries to your *tiles-def-model.xml* file:

```
<definition name="esynListDisplayer.tile" path="/model/esynListDisplayer.jsp" controllerUrl="/initEsy
```

Galaxy

Enable Galaxy export

The following properties are set in the *global.web.properties*. You can override any of those in the *web.properties* file.

```
# in global.web.properties
galaxy.baseurl.default = https://usegalaxy.org
galaxy.url.value = /tool_runner?tool_id=intermine
galaxy.welcomeMessage = <b>Welcome to InterMine, GALAXY users!</b><br/><br/>You can run queries by cl
galaxy.disabledMessage = Galaxy export is disabled.
```

Update Struts config

```
# webapp/src/main/resources/struts-config-model.xml
<action path="/galaxyExportOptions" forward="galaxyExportOptions.page"/>
<action path="/initGalaxyExportOptions"
        type="org.intermine.bio.web.struts.GalaxyExportOptionsController"/>
<action path="/galaxyExportAction" name="galaxyExportForm"
    type="org.intermine.bio.web.struts.GalaxyExportAction" parameter="method"/>

# webapp/src/main/resources/tiles-defs-model.xml
<definition name="galaxyExportOptions.page" extends="layout.template">
    <put name="body" value="galaxyExportOptions.tile"/>
<put name="pageName" value="galaxyExportOptions"/>
</definition>
<definition name="galaxyExportOptions.tile" path="/model/galaxyExportOptions.jsp"
    controllerUrl="/initGalaxyExportOptions.do"/>

# webapp/src/main/resources/struts-config-model-form.xml
<form-bean name="galaxyExportForm" type="org.intermine.bio.web.struts.GalaxyExportForm"/>
```

Customization

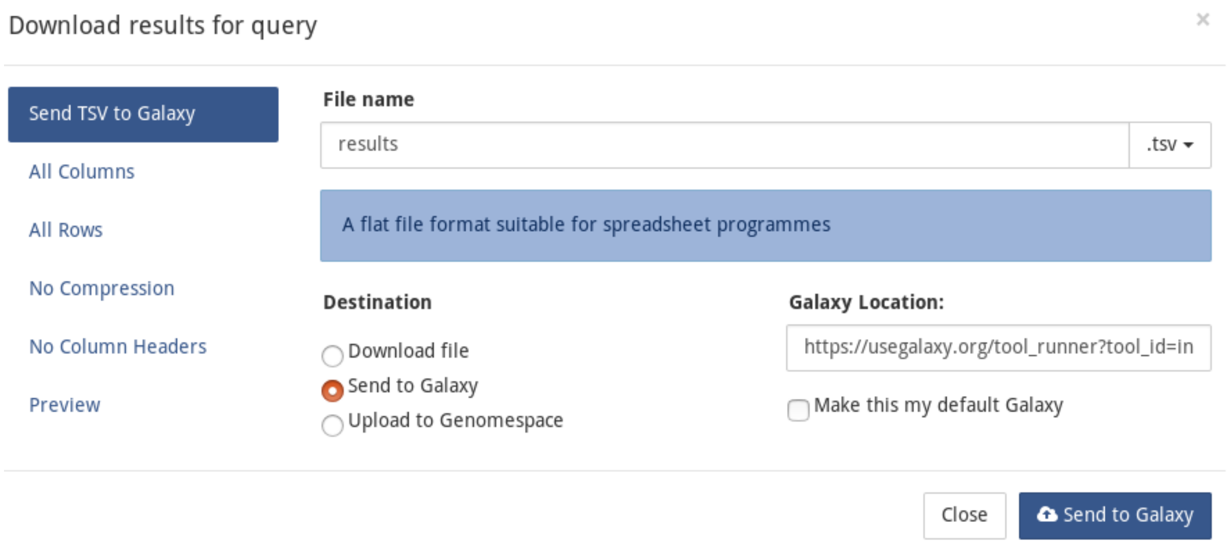
Properties located in the ‘’global.web.properties’‘ file.

parameter	purpose	required?
display	enable Galaxy export	yes[1]_.
disabledMessage	displayed when Galaxy export is disabled	yes
baseurl.default	base url of Galaxy server	yes[2]_.
url.value	tool runner url	yes[3]_.
welcomeMessage	displays on the homepage when coming from Galaxy	yes

```
# galaxy
## set to "false" to disable galaxy
galaxy.display = true
galaxy.disabledMessage = Galaxy export is disabled.
galaxy.baseurl.default = https://usegalaxy.org
galaxy.url.value = /tool_runner?tool_id=intermine
galaxy.welcomeMessage = <b>Welcome to InterMine, GALAXY users</b><br/><br/>You can run queries by \
clicking on the 'Templates' tab at the top of this page.&nbsp;&nbsp;You can send the query results \
to Galaxy from the 'EXPORT' menu in the results page.
```

Export data from InterMine to Galaxy

1. starting from an InterMine instance, e.g. FlyMine, run a query, select the option **Export** -> **Send to Galaxy** and the data will be exported in the galaxy instance specified in the *Galaxy Location* field



Download results for query

Send TSV to Galaxy
All Columns
All Rows
No Compression
No Column Headers
Preview

File name
results .tsv

A flat file format suitable for spreadsheet programmes

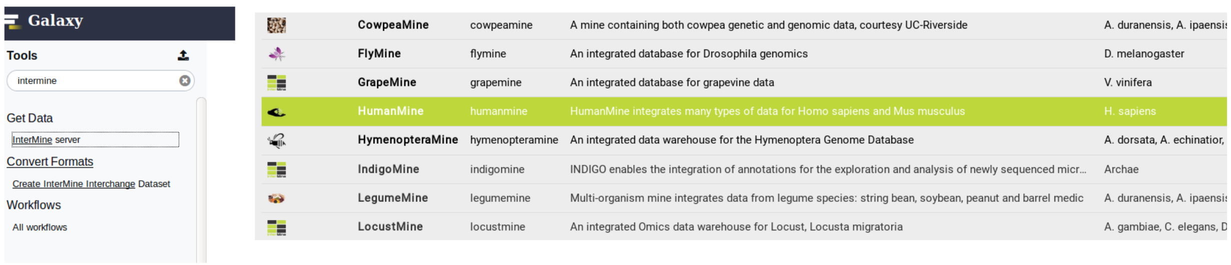
Destination
Download file
[✓ Send to Galaxy]
Upload to Genomespace

Galaxy Location:
https://usegalaxy.org/tool_runner?tool_id=in

Make this my default Galaxy

Close
Send to Galaxy

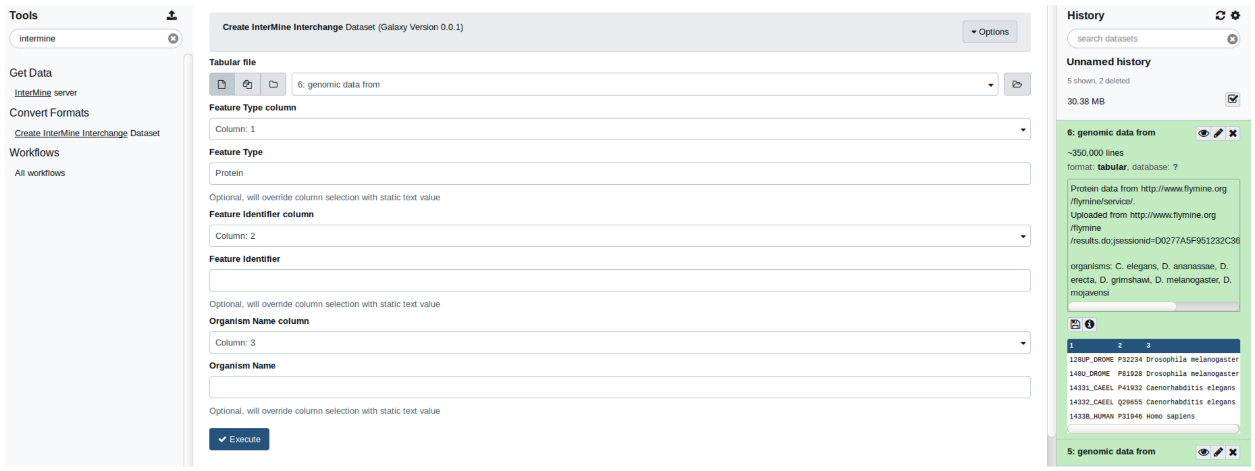
2. starting from Galaxy, use the NEW intermine tool to be redirected to the InterMine registry, select the InterMine instance you want to use to export the data, run the query, select the option **Export** -> **Send to Galaxy** and the data will be exported in the Galaxy instance you started from.



CowpeaMine	cowpeamine	A mine containing both cowpea genetic and genomic data, courtesy UC-Riverside	A. duranensis, A. ipaensis
FlyMine	flymine	An integrated database for Drosophila genomics	D. melanogaster
GrapeMine	grapemine	An integrated database for grapevine data	V. vinifera
HumanMine	humanmine	HumanMine integrates many types of data for Homo sapiens and Mus musculus	H. sapiens
HymenopteraMine	hymenopteramine	An integrated data warehouse for the Hymenoptera Genome Database	A. dorsata, A. echinatior,
IndigoMine	indigomine	INDIGO enables the integration of annotations for the exploration and analysis of newly sequenced micr...	Archae
LegumeMine	legumemine	Multi-organism mine integrates data from legume species: string bean, soybean, peanut and barrel medic	A. duranensis, A. ipaensis
LocustMine	locustmine	An integrated Omics data warehouse for Locust, Locusta migratoria	A. gambiae, C. elegans, D

Export identifiers from Galaxy to InterMine

Use the new **InterMine interchange dataset** to generate an intermediate file (tsv formatted)



Create InterMine Interchange Dataset (Galaxy Version 0.0.1)

Tabular file
6: genomic data from

Feature Type column
Column: 1

Feature Type
Protein

Optional, will override column selection with static text value

Feature Identifier column
Column: 2

Feature Identifier

Optional, will override column selection with static text value

Organism Name column
Column: 3

Organism Name

Optional, will override column selection with static text value

Unnamed history
5 shown, 2 deleted
30.38 MB

6: genomic data from
~350,000 lines
format: **tabular**, database: **?**

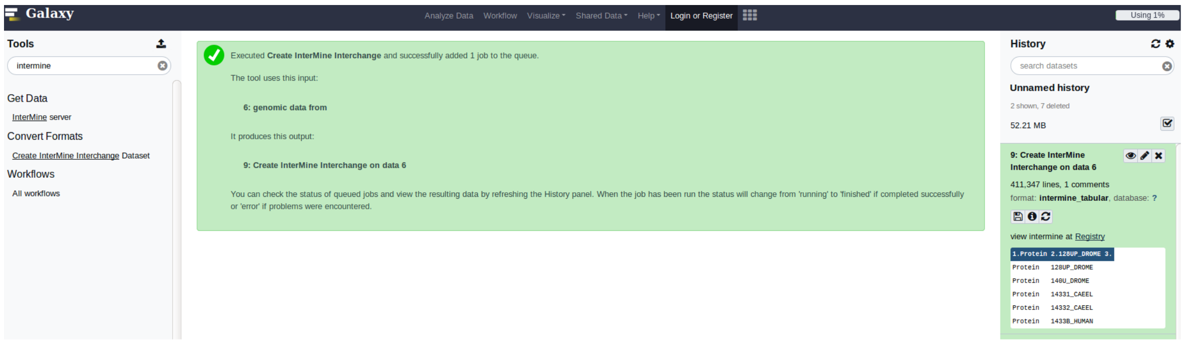
Protein data from http://www.flymine.org
/flymine/service/.
Uploaded from http://www.flymine.org
/flymine
/results.do;jsessionid=D0277A5F951232C36

organisms: C. elegans, D. ananassae, D. erecta, D. grimshawi, D. melanogaster, D. mojavensi

```
1           2      3
128UP_DROME P32234 Drosophila melanogaster
140U_DROME  P81928 Drosophila melanogaster
14331_CAEEL P41932 Caenorhabditis elegans
14332_CAEEL Q20655 Caenorhabditis elegans
1433B_HUMAN P31946 Homo sapiens
```

5: genomic data from

and then click on *View InterMine at Registry* to be redirected to the InterMine registry in order to chose the InterMine instance to export the identifiers to.



Executed **Create InterMine Interchange** and successfully added 1 job to the queue.

The tool uses this input:

6: genomic data from

It produces this output:

9: Create InterMine Interchange on data 6

You can check the status of queued jobs and view the resulting data by refreshing the History panel. When the job has been run the status will change from 'running' to 'finished' if completed successfully or 'error' if problems were encountered.

Unnamed history
2 shown, 7 deleted
52.21 MB

9: Create InterMine Interchange on data 6
411,347 lines, 1 comments
format: **intermine_tabular**, database: **?**

view intermine at Registry

```
1.Protein 2.128UP_DROME 3.
Protein   128UP_DROME
Protein   140U_DROME
Protein   14331_CAEEL
Protein   14332_CAEEL
Protein   1433B_HUMAN
```

GBrowse

You can link out to an external GBrowse instance. See here for an example: http://intermine.readthedocs.org/en/latest/webapp/report-page/report-displayers-examples/#gbrowse

If you would like to host your own genome browser using InterMine data, see *JBrowse*

Heatmap

InterMine makes use of canvasXpress heatmap to visualize gene expression data.

CanvasXpress is a javascript library based on the <*canvas*> tag implemented in HTML5. It is written by Isaac Neuhausi.

Hierarchical and k-Means clustering algorithms and zoom in/out functionality have been implemented within the heatmap.

An example in modMine

A specific heatmap application can be referred in modMine. It visualizes fly expression data (example) generated from modENCODE project.

The raw data is parsed and converted to InterMine objects. In a Struts controller, the expression data will be fetched by running a InterMine path query and parsed to JSON string. The JSON string will be sent to a JSP page by a http request to feed into heatmap.

Expression data source FlyExpressionScoreConverter is a specific data converter for modENCODE fly expression data. The class is located at *bio/sources/modmine/fly-expression-score*. Any other similar expression data conversion tasks can take the data source as a reference.

Exprssion data type is an extension of InterMine core model. It is addressed in *modmine/dbmodel/resources/modencode-metadata_additions.xml*

```
# modmine/dbmodel/resources/modencode-metadata_additions.xml
<class name="GeneExpressionScore" is-interface="true">
    <attribute name="score" type="java.lang.Double" />
    <reference name="gene" referenced-type="Gene" reverse-reference="expressionScores" />
    <reference name="cellLine" referenced-type="CellLine" />
    <reference name="developmentalStage" referenced-type="DevelopmentalStage" />
    <reference name="submission" referenced-type="Submission" />
    <reference name="organism" referenced-type="Organism" />
</class>
```

A better practice would be to add the model extension to a source specific additions.xml under a source directory.

Controller The controller class HeatMapController is a component of Struts MVC framework. It holds the logic to process user requests, and seletcs a proper wabpage to user.

In HeatMapController, a query is run to fetch expression scores from database (ref method *queryExpressionScore*), then the results are parsed to JSON string (ref method *getJSONString*) and set in the request (ref method *findExpression*).

Struts config:

```
# modmine/webapp/resources/struts-config-model.xml
<action path="/initHeatMap"
        type="org.modmine.web.HeatMapController" />

<action path="/heatMap" forward="heatMap.page" />


#  modmine/webapp/resources/tiles-defs-model.xml
<definition name="heatMap.tile" path="/model/heatMap.jsp"
        controllerUrl="/initHeatMap.do"/>

<definition name="heatMap.page" extends="layout.template">
        <put name="body" value="heatMap.tile"/>
        <put name="pageName" value="heatMap"/>
</definition>
```

Web page heatMap.jsp displays heatmap. canvasXpress object takes expression JSON string and other parameters in to create a heatmap (in modMine, we have two separate heatmaps for cell line and developmental stage respectively). jQuery was used to adjust page layout.

Further development

A modern way of creating widget like heatmap would be using InterMine webservices framework to query and generate JSON strings and embed heatmap on any web page. To learn more...

An alternative library would be D3.js, an example of heatmap can be found here. In ThaleMine there is a D3 implementation (see any gene list report page, for example, code. However canvasXpress is particular designed to display genomics data, D3 is for a broader use.

JBrowse

InterMine 1.3.1 supports the JBrowse REST web-service specification (see configuring JBrowse) which means that you can run a JBrowse installation directly off the InterMine web-services.

This documentation has been tested with JBrowse-1.16.4.

Build Your InterMine Database

If you want to be able to have a hierarchical view of your features on JBrowse add this to the <*post-processing*> section of your project XML file and then build your database:

```
<post-process name="populate-child-features"/>
```

See *Post processing* for details.

Install JBrowse

You will need an installation of JBrowse for this task. Instructions on doing this can be found at installing JBrowse.

Note: you need to set

```
<div class="jbrowse" id="GenomeBrowser" data-config='"allowCrossOriginDataRoot": true'>
```

in the index.html file of your JBrowse installation.

Add JBrowse to InterMine

Add JBrowse to your report pages by adding this entry to your webconfig-model.xml file:

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.JBrowseDisplayer"
                 jspName="model/jbrowseDisplayer.jsp"
                 replacesFields=""
                 placement="Genomics"
                 types="SequenceFeature"/>
```

See *Report Displayers Examples* for more information.

Add the location of your JBrowse installation to your *web.properties* or mine properties file, for example:

```
jbrowse.install.url = http://jbrowse.intermine.org
```

Point JBrowse at your InterMine

Add your new mine-based dataset to your configuration file. For example to add *D. melanogaster* data from FlyMine as a JBrowse dataset, the following configuration in *jbrowse_conf.json* would suffice:

```
{
  "datasets": {
     "FlyMine-7227": {
      "url": "?data=http://www.flymine.org/query/service/jbrowse/config/7227",
      "name": "FlyMine"
    },
    ...
  }
}
```

Once in place, you can visit your JBrowse *index.html* and see the data from FlyMine.

Configuring InterMine's JBrowse integration

By default, all InterMine classes that inherit from the SequenceFeature model class will have tracks.

However, this can be inappropriate since some of those classes may not have data.

You can make entries in *web.properties* to configure which tracks appear. For instance, if you just want Gene, CDS, UTR and Promoter tracks then in *<mine>/webapp/resources/web.properties* configure

```
org.intermine.webservice.server.jbrowse.genomic.track.Gene.class=Gene
org.intermine.webservice.server.jbrowse.genomic.track.CDS.class=CDS
org.intermine.webservice.server.jbrowse.genomic.track.UTR.class=UTR
org.intermine.webservice.server.jbrowse.genomic.track.Promoter.class=Promoter
```

Here, track names are the first component of the key after org.intermine.webservice.server.jbrowse.genomic.track (e.g. Gene on the first line). These track names are used to group related properties and are not used in JBrowse display. The rest of the key name (here always class) specifies the InterMine class to be used for this track. <div class="jbrowse" id="GenomeBrowser" data-config="'"allowCrossOriginDataRoot": true'> JBrowse parameters can also be set for individual tracks within InterMine. For instance, in *web.properties*, if one wanted to give all 4 of the tracks defined above different colours then one would set

```
org.intermine.webservice.server.jbrowse.genomic.track.Gene.style.color=red
org.intermine.webservice.server.jbrowse.genomic.track.CDS.style.color=yellow
org.intermine.webservice.server.jbrowse.genomic.track.UTR.style.color=green
org.intermine.webservice.server.jbrowse.genomic.track.Promoter.style.color=blue
```

For the full list of properties, please see the canvas section of the JBrowse Configuration Guide.

1.6.17 Monitoring Site Usage

Google Analytics

To enable Google Analytics to track usage of your webapp you need to set up your Analytics account and get a '''code'' from Google then add a property to your *.intermine/xxxmine.properties* file:

```
google.analytics.id=CODE
```

This places the Google javascript to track usage views to every page of the webapp.

To modify the message that is going to be displayed to the user asking if they agree to the usage of cookies modify:

```
google.analytics.message=I accept cookies from this site
```

If no key - message is provided, no message is shown and cookies are accepted by default.

InterMine User Tracking

The following user activities are recorded during browsing of all mine webapps, to track the usage:

- template query executions
- query executions from the QueryBuilder
- list creations by upload, operations (copy, union...) or from result tables
- list executions
- searches by keyword
- user logins

The trackers, recording all tracks, are defined in the *global.web.properties* file under the property *webapp.trackers*.

All tracks are saved into specific tables, created automatically, if they don't already exist, in the userprofile database.

The tables are:

- templatetrack
- querytrack
- listtrack
- searchtrack
- logintrack

The table are created at the startup of the webapp.

If some table definitions needed to be updated, the browsing of the webapp is blocked, and a warning message is displayed.

Run the ant task *update-templatetrack-table* in the webapp directory and then restart the webapp.

Usage page in the webapp

You can access to the *Usage* page, via the MyMine page, only if you are a superuser.

The page shows all tracks grouped in these sections: Template usage, Custom queries executed, Logins, Keyword Searches, List Analysis page views and List Creation.

Each section contains:

- a diagram showing the trend of that specific track during the time range selected: 1d, 5d, 1m, 3m, 6m, 1y. The diagrams point out the number of tracks per day (number of template query execution, number the query execution....) without specify wich template or type query(Gene, Protein...) has been executed.
- a table showing the number of tracks for each template, type of query, type of list....

Furthemore, in the ‘’Template usage” section, there is a pie diagram showing the first 9 most popular templates and their number of executions in the time range selected: last 2 weeks, last month, last 3 months, last year.

Under the label ‘Other’ the number of executions of templates from 10th to 15th position.

Search engines

This document discusses the relationship between your InterMine-based website and search engines.

If you launch your website, eventually your site will be found and indexed by Google or other search engines.

Being listed on the search engines is beneficial as it will drive traffic to your site. However being listed can result in unintended consequences, like exposing “hidden” parts of your site. InterMine provides an easy way to control which parts of the website are indexed by the search engines.

Search Engine Optimisation

To use each of the search engines’ webmaster tools, you need to include a *CODE* in a meta tag on your website. You can do this by updating your properties file like so, replacing *CODE* with the value that Google/Microsoft/Yahoo provide:

```
# MINE.properties

# http://www.google.com/analytics
google.analytics.id=CODE

# http://www.google.com/webmasters
searchengines.google=CODE

# http://www.bing.com/webmaster
searchengines.msn=CODE
```

See also:

Google Analytics

“‘robots.txt’“

The easiest way to control what the search engines index is to use a file called robots.txt. Robots use this file to determine which parts of the site they can visit. This file should be located in the root of your site, ie. www.flymine.org/robots.txt

You can also specify which search engines can index your site, e.g. Google or Yahoo. Here is an example file:

```
Sitemap: sitemap_index.xml

User-agent: *
Disallow: /

User-agent: Googlebot
Disallow:
Disallow: /release-8.2/
Disallow: /release-8.1/

User-agent: Slurp
Disallow:
```

```
Disallow: /release-8.2/
Disallow: /release-8.1/

User-agent: msnbot
Disallow:
Disallow: /release-8.2/
Disallow: /release-8.1/
```

This file bans all search engine robots except for Google, Yahoo, and MSN. In addition this file forbids the robots to index files in the release-8.1 and release-8.2 directories.

Read more about this document on the http://www.robotstxt.org website.

NOFOLLOW

You can restrict access to directories via the robots.txt file, but you can also configure your site to allow or forbid access to specific web pages.

To prevent the search engine robots from following links on that page, set the noFollow attribute in the InterMineWebApp.properties file:

```
# MYMINE
mymine.title = MyMine
mymine.description = Your list of saved lists and queries
mymine.tab = mymine
mymine.noFollow = true
```

Sitemaps

Search engines often have difficulty indexing dynamic websites. The easiest solution for this is provide a sitemap that indicates which pages should be indexed.

1.6.18 Website Admin

The SuperUser is the administrator of your InterMine webapp. The SuperUser can use tagging to configure the appearance and functionality of the webapp.

The SuperUser account is created when the UserProfile database is built using the properties specified in the *InterMine properties* file .

Templates

All logged in users can create template queries, but the SuperUser can make them available to all users by tagging them as public templates. Making a template query is an easy way to get users of your webapp to the data they want very quickly.

Tagging

Template queries and lists

The SuperUser can change where templates and lists appear by adding tags via the templates and lists pages in the MyMine section of the webapp. Only the administrator can apply/view/edit tags starting at *im:* The tag data is stored in the user-profile database.

tag	purpose
im:public [16]	make list/template viewable by all users
im:frontpage	put list on home page
im:converter	template used in generating links in the 'Convert' section on the list analysis page
im:aspect:CategoryName	template appears underneath specified category. For instance template with im:aspect:Genomics tag will be displayed in Genomics category on the report page and on the home page
im:report	allows template to be displayed on report or list analysis page
im:order:n	specify the order lists should go in (on homepage only currently). If two lists have the same Integer "n" value, natural ordering on the list name will be applied as a decisive criterion

Fields and collections

The SuperUser can change how fields are displayed by adding tags via the report page.

tag	purpose
im:hidden	hides the field/collection
im:summary	add collection to 'Summary' section of report page
im:aspect:CategoryName	collection appears underneath category

Classes

The SuperUser can change how classes are displayed by adding tags via the model browser.

tag	purpose
im:aspect:CategoryName	class appears on aspect page
im:preferredBagType	class appears first in the class selection

im:converter tag

If a template is tagged with *im:converter*, it is:

1. Used by the list analysis page, in the "Convert" section.
2. Used by the list upload page to converter between types.

- Eg, the user pastes in a protein identifier, but chooses "Gene" from the type drop down menu. A converter template can be used to look up the *Gene* corresponding to the given *Protein*.

To work as a converter the template must follow the following pattern:

- the top-level class in the query must be the class we wish to convert *from* (eg. *Gene*)
- there must be exactly one editable constraint - the *id* field of the top level class (eg. *Gene.id*)
- the fields selected for output must be *Gene.id* and the id field of the class to convert *to*

[16] Editable by all admins

Normally the *id* field isn't shown in the query builder and probably isn't useful in other queries. Only the administrator user can create queries using the *id* field. Here is an example converter template:

```
<template name="Gene_To_Protein_Type_Converter" title="Gene to protein type converter" longDescriptio
        <query name="Gene_To_Protein_Type_Converter" model="genomic" view="Gene.id Gene.proteins.id"
        <node path="Gene" type="Gene"></node>
        <node path="Gene.id" type="Integer">
                <constraint op="=" value="0" description="Gene.id" identifier="Gene.id" editable="tru
        </node>
        </query>
</template>
```

1.6.19 User Accounts

Userprofile

The user profile is an InterMine ObjectStore which stores Profile information such as username and password, tags, queries, lists and templates.

Creating a new UserProfile database

First you must create the empty database in Postgres.

```
# create the new empty database
~/git/flymine $  createdb userprofile-biotestmine
```

These commands are needed in the webapp to initialise a userprofile database:

```
# this comment populates superuser, default templates etc.
~/git/flymine $ ./gradlew buildUserDB
```

Releasing a webapp with a new production database

If you already have a userprofile database and want to keep the data it contains, you can do this:

1. Verify that the *serialNumber* in the new production db and in the userprofile are different. Only in this case, the upgrading list process updates the lists when the user logs in

```
# run in production and userprofile database.  when releasing a new product
select * from intermine_metadata where key='serialNumber';
```

2. Release the webapp pointing to the new production db.
3. In the *savedbag* table the field *intermine_state* should be set to *false*.
4. When the user logs in, the upgrading list process will update the list (using *bagvalues* table)

- if there are no conflicts the flag will be set to *true* and the user will not have to take any action
- if there are issues (eg. if a gene has merged with another) the flag will be set to *false*, and the user will have to manually upgrade their list.

Templates and tags

Default templates and tags are defined in *default-template-queries.xml*.

These are loaded when you build a userprofile database.

Back ups

For our mines, we have a script to back up the user databases every five minutes, but only if there has been a change.

Open ID

InterMine web-applications allow users to create accounts and sign in to these accounts by authenticating with a selection of Open-ID providers, including Yahoo.

To sign in with one of these authentication providers: 1. Click on ‘‘login‘‘ (in the upper-right). 2. Click the name of the Open-ID provider you wish to use. 3. Authenticate yourself with your provider. 4. You will be redirected to your mine when finished.

Note: Google has shut down its OpenID-2 service. To continue using Google authentication you must use OAuth2 authentication! See the section on editing web-properties for more details.

To set this up for a mine you administer:

- The most important thing is to set up a couple of properties correctly in your mine’s properties file (located in the *.intermine* directory), eg:

```
webapp.baseurl=http://beta.flymine.org
webapp.path=intermine-test
```

If you do not wish to allow Open-ID accounts, set the property “openid.allowed=false” in any of the property files that end up in the WEB_PROPERTIES map.

1.6.20 Performance

InterMine web-applications rely on a server to deliver static files such as JavaScript and CSS. The default location for this server is “http://cdn.intermine.org”. Installing your own CDN may increase web site performance.

Setting up your own Content Delivery Network

This dependency is easy to remove. You can host all these files yourself from any location. We recommend doing the following:

- Cloning your own copy of the CDN This means you have local copies of all the files.
- Making the root directory of your checkout visible through a web-server (an Apache ‘alias’ directive is sufficient). These resources should be accessible through CORS enabled web-servers - see: http://enable-cors.org
- Change the value of the ‘head.cdn.location’ property in your web-app. This is currently configured in ‘global.web.properties’ as *head.cdn.location = http://cdn.intermine.org*
- Supply the location of your CDN at runtime to JavaScript components that may use it: Set the option “CDN.server” to the appropriate URL (see http://intermine.readthedocs.org/en/latest/webapp/properties/javascript-options/)

1.6.21 Diagnostic

Occasionally something may go wrong with your webapp - your webapp may fail to load in your browser, not reflect your most recent changes and so on. In our experience, following the steps listed here should fix ~99% of any problems you encounter.

Restart Tomcat

Restarting Tomcat may fix your issue. If you find you have to restart Tomcat often, you may want to give Tomcat more memory.

Also, if in a deadlock, Tomcat may not shutdown successfully. Be sure to check the Tomcat process really is gone before starting a new one.

Verify MINE.properties file

The *base-url* property must valid or else queries will not run properly.

This file must live in the *.intermine* directory.

Verify Tomcat config

Please make sure you have configured Tomcat correctly. See *Tomcat*

Force recompile

Run this command in your *webapp* directory:

```
$ ./gradlew clean
```

Verify */build* is gone from your *webapp* directory.

Re-release webapp

```
$ ./gradlew cargoReDeployRemote
```

1.6.22 Building Javadoc

Package-specific Javadoc

Dependency note

Note that package-specific Javadoc generation only works if you have successfully built the package first. If you haven't built it before, you're likely to get error messages about missing files.

Building

To generate Javadoc at a package-specific level, change directory to the webapp directory of a given mine, and run *ant javadoc*. Assuming you're at the root of your intermine directory:

```
~/git/flymine $ ./gradlew javadoc
```

Upon successful build, you'll be able to find the Javadoc under the *build/javadoc* folder. For the FlyMine example above, it'd be at *flymine/webapp/build/javadoc*.

Intermine Javadoc

If you just want to browse the docs, you can see the most recent version at http://intermine.org/intermine/.

Using Travis to auto-deploy Javadoc to GitHub Pages

If you have your InterMine repo set up to automatically run tests using Travis, you can deploy the documentation automatically whenever new code is checked into your *master* branch, using Github Pages as a host.

Prerequisites

In order to deploy, there must be an encrypted *$GH_TOKEN* set in *.travis.yml* to authenticate with Github. This is safe because of the way Travis treats encrypted variables. This token needs to be generated by someone with access to the repo

Generating a token Go to the Personal access tokens section of Github, and create a now token with *repo* permissions only. If it's a public repo, then *public_repo* permissions will suffice. Name it something memorable, and copy it down somewhere safe when you are shown it, as you're only shown it the once.

Encrypting the token **Important: to ensure you don't inadvertently leak your token, granting someone else write-access to your repo, you must encrypt this token! Do *not* paste it into .travis.yml without encrypting it first!** To encrypt you Github token, you're need to install Travis CLI locally if you haven't already, then run

```
$ travis encrypt GH_TOKEN=put-your-very-secret-github-token-here
```

This will output the encrypted variable into your console, looking something like:

```
secure: long string of nonsense here
```

Copy the entire output to your *.travis.yml* under the env global section. You should end up with a section like this. It's ok to have more than one secure environment variable.

```
env:
  global:
  - secure: that same long string of nonsense
```

See Travis's documentation on encrypting environment variables for more information.

Assuming the repo is already set up to be tested in Travis, this should be all you need to set up automatic deployments

Configuring Travis to auto-deploy Javadoc from branches other than master If you wish to deploy javadoc from a different branch, perhaps a development / test branch, find this line in your *.travis.yml*

```
after_success:
  - test $TRAVIS_BRANCH == "master" && bash config/travis/deploydocs.sh
```

Simply change the value of "master" to the branch you wish to use.

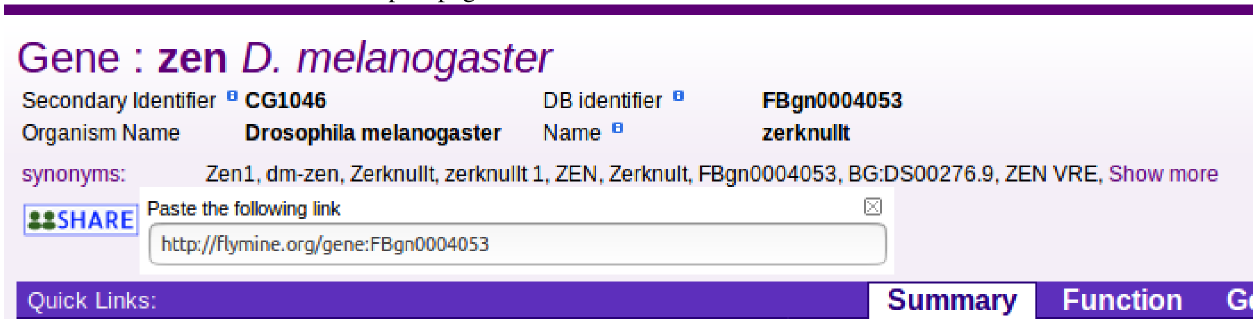
1.6.23 Permanent URLs

InterMine generates stable and unique URLs to identify the report pages for biological entities.

They are based on class names combined with local IDs provided by the data resource providers and therefore they are persistent.

In FlyMine, for example, the URL of the report page for the gene zen, with primary identifier P19107, will be http://flymine.org/gene:FBgn0004053.

These URLs are used to share the report page with other users.


Gene : zen D. melanogaster

Secondary Identifier CG1046 DB identifier FBgn0004053

Organism Name Drosophila melanogaster Name zerknullt

synonyms: Zen1, dm-zen, Zerknullt, zerknullt 1, ZEN, Zerknult, FBgn0004053, BG:DS00276.9, ZEN VRE, Show more

Paste the following link

http://flymine.org/gene:FBgn0004053

The *class_keys.properties* file specifies the keys used to generate the permanent URLs. If not specified, the primaryidentifier key is used.

The format is:

# class_keys.properties
<CLASSNAME>_URI <FIELDNAME>

The classes and field names are case sensitive.

For example:

key	value
Pathway_URI	identifier

No need to specify the keys for the classes defined in the core model (e.g. protein, organism, publication...).

See *Class keys* for details about this file.

1.6.24 Web pages markup

We have applied structured data in JSON-LD format to InterMine web pages (using Bioschemas.org types and profiles), to improve findability so search engines can give more relevant results to users.

The markup are disabled by default, to enable them set the property *markup.webpages.enable* to *true* in the *web.properties file*.

We have applied the following markup:

Type	Applicable
DataCatalog	Main Home Page
DataSet	Report page for entitites with type DataSet

Home page markup

property	description	example
identifier	The identifier for the mine instance, based on the namespace assigned in the intermine registry [17]	https://registry.intermine.org/flymine
name	The name of the InterMine instance	FlyMine
descrition	The description of the InterMine instance	An integrated database for Drosophila and Anopheles genomics
url	The url of the InterMine instance	http://flymine.org
dataset	The list of the datasets stored in the InterMine instance containing name and url	

Report page markup for DataSet

property	description	example
name	The name of the dataset	FlyAtlas
description	The description of the dataset	Affymetrix microarray-based atlas
identifier	The url of the dataset, if provided, or the permanent URL of the report page	http://www.flyatlas.org/
url	The permanent URL of the report page	http://flymine.org/flymine/dataset:flyatlas

1.6.25 Customising the default queries in your io-docs application

You can have default queries defined for your iodocs application documenting the *Web Services* available in InterMine, see http://iodocs.apps.intermine.org

To set your mine default query for the *'query/results'* service of your mine, add it to your web.properties configuration file, e.g.

add to `webapp/src/main/webapp/WEB-INF/web.properties`

```
services.defaults.query = <query model="genomic" view="Gene.secondaryIdentifier Gene.symbol Gene.pri
```

1.6.26 Overwrite any JSP

When the webapp is compiled, the order of projects is:

1. intermine/webapp

[17] When an InterMine instance is added to the registry, an unique and persistent namespace is assigned by the administrator. Some examples of namespaces: flymine, humanmine, flymine.beta. The identifier will be: https://registry.intermine.org/{namespace}. These identifiers are actionable, so if you put https://registry.intermine.org/{namespace} in the address bar of your browser, you will be redirected to URL set in the registry for the FlyMine. If the InterMine instance is not register, the url will be used instead.

2. bio/webapp <– overwrites files in intermine/webapp
3. MINE_NAME/webapp <– overwrites files in intermine/webapp and bio/webapp

You can overwrite any JSP in the intermine or bio/webapp projects by having a JSP of the same name in your mine's webapp directory. The danger of this is that you will have to upgrade these JSPs manually.

1.7 Web Services

InterMine provides programmatic access to its features via web services. This allows users to automate:

- Data retrieval (custom queries, templated queries, keyword searches).
- List creation/analysis/management
- User profile management
- Data-model introspection

For a full listing of web service capabilities on various mines please see the HTTP API documentation.

1.7.1 Getting Started

Install Required Dependencies If you are reading this page, we make the assumption you know how to write and run programs in your language of choice. You will probably want to download and install the appropriate client library (see API and Client Libraries), which typically involves the standard package manager for the given platform.

Look at some example code We assume you are already familiar with the InterMine web interface, as provided by sites such as FlyMine. Each result table in the web interface includes a mechanism for generating code using one of the client libraries which generates the same results as those seen in the table (click on the **code** button). The generated code is meant to help get you started with the use of the client libraries.

There is also a `/web-services/tutorial` for the Python API.

Modify the code so it does what you want Working from the generated stub, you can edit the code to perform your intended task. You will probably want to refer to the API documentation for your target language (see below).

1.7.2 API and Client Libraries

InterMine exposes its functionality over an HTTP API (generally following RESTful principles, but there is a bit of RPC there). Client libraries are available in commonly used languages so you can get started quickly. All our code is hosted on Github, and contributions are welcome. All InterMine code is free and open-source, released under the LGPL (see *Legal*).

For information on the underlying API, and the supported libraries, please visit the following links:

HTTP API Documentation on services available from mines: http://iodocs.apps.intermine.org

Java Download | API | Tutorial | Source

Perl Client Download | API | Tutorial | Source

Python Client Download | API | Tutorial | Source

Ruby Client Download | API | Tutorial | Source

JavaScript Client (for Bowser and node.js) Download | API | Tutorial | Source

R Client Download | Docs | Tutorial | Source

1.7.3 Authentication

Authenticated web services are accessed via tokens: either 24-hour anonymous tokens or permanent user API key tokens. See `/web-services/authentication`

1.8 Embedding InterMine components

The following are libraries that compile to JavaScript which can be embedded on *any* webpage.

1.8.1 List Widgets

List Widgets Questions & Answers

Source files

Source files for the List widgets client.

Using a temporary list on the fly

Requirements

1. InterMine Generic WebService Client Library from GitHub or InterMine CDN.
2. InterMine List Widgets Client Library from GitHub or InterMine CDN.
3. A mine that has the desired Enrichment Widget configured.
4. An API Access Key generated by logging in to MyMine and visiting the API Key tab, then clicking on Generate a new API key. This assumes that you do not want to automatically provide the API key as is the case of within mine embedding that can be seen for example here.

Code First require the JavaScript libraries needed to run the example. You probably have your own version of a Twitter Bootstrap compatible CSS style included on the page already.

```
<!-- dependencies -->
<script src="http://cdn.intermine.org/js/jquery/1.9.1/jquery-1.9.1.min.js"></script>
<script src="http://cdn.intermine.org/js/underscore.js/1.3.3/underscore-min.js"></script>
<script src="http://cdn.intermine.org/js/backbone.js/0.9.2/backbone-min.js"></script>

<!-- intermine -->
<script src="http://cdn.intermine.org/api"></script>
<script src="http://cdn.intermine.org/js/intermine/imjs/latest/im.js"></script>
<script src="http://cdn.intermine.org/js/intermine/widgets/latest/intermine.widgets.js"></script>
```

The next step is defining a couple of variables.

```
var root = 'http://www.flymine.org/query';
var tokn = 'U1p3r9Jb95r2Efrbu1P1CdfvKeF'; // API token
var name = 'temp-list-from-js-query'; // temporary list name
```

Now we connect with the mine through *InterMine JavaScript Library*.

```
// Service connection.
var flymine = new intermine.Service({
    'root':  root,
    'token': tokn
});
```

Then we define the query whose results will be converted into a list later on.

```
// The query herself.
var query = {
    'select': [ 'symbol', 'primaryIdentifier' ],
    'from': 'Gene',
    'where': {
        'symbol': {
            'contains': 'ze'
        }
    },
    'limit': 10
};
```

Now we call the mine converting the results of the query into a list.

```
flymine.query(query)
       .then(function madeQuery (q) {
         // q is an instance of intermine.Query.
         return q.saveAsList({'name': name}); })
       .then(function savedList (list) {
         // list is an instance of intermine.List.
         console.log(list.size); });
       .fail(function onError (error) {
         console.error("Something went wrong");});
```

Now, in the function *savedList*, we can instantiate the List Widgets client and display the result.

```
var widgets = new intermine.widgets(root + '/service/', tokn);
// A new Chart List Widget for a particular list in the target #widget.
widgets.chart('flyfish', name, '#widget');
```

The only problem with this approach is that if we make this sort of call multiple times, we will fail on the second and subsequent ocassions as we will get a WebService exception telling us that the 'temporary' list name is taken. *Thus inspect the code of the example to see how to make a call to the service to delete/reuse the list if it exists.*

Defining custom actions on widget events

In a mine context, List Widgets are configured automatically to e.g. display a *Query Results* when clicking on "Create a List".

Outside of a mine context, one needs to pass in what happens when one interacts with the Widgets. You can also decide whether to show/hide either/and/or title or description of the widget (for everything else use CSS).

Clicking on an individual match (Gene, Protein etc.) in popover window:

```
var options = {
    matchCb: function(id, type) {
        window.open(mineURL + "/portal.do?class=" + type + "&externalids=" + id);
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Clicking on View results button in a popover window:

```
var options = {
    resultsCb: function(pq) {
        // ...
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Clicking on Create list button in a popover window:

```
var options = {
    listCb: function(pq) {
        // ...
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

I want to hide the title or description of a widget.

```
var options = {
    "title": false,
    "description": false
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Showing a Results Table

The example below assumes that you have resolved all *Query Results* dependencies and have a PathQuery in JSON/JavaScript format that you want to display in a `#container`:

```
// Define a query as above
var pq = {from: "Gene", select: ["symbol", "organism.name"], where: {Gene: {in: "my-list"}}};
// use an instance of a Service or perhaps you already have one.
var service = new intermine.Service({'root': service, 'token': token});
// Create a new ResultsTable.
var view = new intermine.query.results.CompactView(service, pq);
// Say where to put it.
view.$el.appendTo("#container");
// Show it.
view.render();
```

List enrichment widgets statistics

Enrichment widgets are located on the list analysis page. There are a number of different types of enrichment widgets, but all list a term, a count and an associated p-value. The term can be something like a publication name or a GO term. The count is the number of times that term appears for objects in your list. The p-value is the probability that result occurs by chance, thus a lower p-value indicates greater enrichment.

Method

The p-value is calculated using the Hypergeometric distribution. Four numbers are used to calculate each p-value:

```
      (M choose k) (N-M choose n-k)
P =   -----------------------------
              N choose n
```

n the number of objects in your list

N the number of objects in the reference population

k the number of objects annotated with this item in your list

M the number of objects annotated with item in the reference population

Apache library - Hypergeometric Distribution

Multiple Test Correction

When multiple tests (statistical inferences)are run in parallel, the probability of false positive (Type I) errors increases. To address this issue, many multiple test corrections have been developed to take into account the number of tests being carried out and to correct the p-values accordingly. Enrichment widgets have three different multiple test corrections: Bonferroni, Holm-Bonferroni, and Benjamini Hochberg.

In enrichment widgets the number of "tests run" is the number of terms associated with objects in the "reference list". Please Note, in earlier versions of InterMine (0.95 and below) the number of "tests run" was the number of terms associated with objects in the "query list". This change has made the multiple test correction more rigorous, and will reduce the occurrence of spuriously low p-values.

Each enrichment widget has four test correction options:

None No test correction performed, these are the raw results. These p-values will be lower (more significant) than if test correction was applied.

Bonferroni Bonferroni is the simplest and most conservative method of multiple test correction. The number of tests run (the number of terms associated with objects in the reference list) is multiplied by the un-corrected p-value of each term to give the corrected p-value.

Holm-Bonferroni

```
Adjusted p-value = p-value x (number of tests - rank)
```

Benjamini Hochberg This correction is the less stringent than the Bonferroni, and therefore tolerates more false positives.

```
Adjusted p-value = p-value x (number of tests/rank)
```

1. The p-values of each gene are ranked from the smallest to largest.
2. The p-value is multiplied by the total number of tests divided by its rank.

Gene length correction The probability of a given set of genes being hit in a ChIP experiment is amongst other things proportional to their length – very long genes are much more likely to be randomly hit than very short genes are. This is an issue for some widgets – for example, if a given GO term (such as gene expression regulation) is associated with very long genes in general, these will be much more likely to be hit in a ChIP experiment than the ones belonging to a GO term with very short genes on average. The p-values should be scaled accordingly to take this into account. There are a number of different implementations of corrections, we have choosen the simplest one. The algorithm was developed by Taher and Ovcharenko (2009) for correcting GO enrichment. Corrected probability of observing a given GO term is equal to the original GO probability times the correction coefficient CCGO defined for each GO term.

```
Adjusted P = P x CCGO
```

where the correction coefficient CCGO is calculated as:

```
           LGO/LWH
CCGO = ----------------
           NGO/NWG
```

LGO Average gene length of genes associated with a GO term

LWG Average length of the genes in the whole genome

NGO Number of genes in the genome associated with this GO term

NWG Total number of genes in the whole genome.

Note: The relevant InterMine source.

Reference population The reference population is by default the collection of **all the genes with annotation** for the given organism. This can be changed to any available list of genes.

References

GOstat: Find statistically overrepresented Gene Ontologies within a group of genes
Beissbarth T, Speed TP.
Bioinformatics. 6.2004; 20(9): 1464-1465.
PubMed id: 14962934

GO::TermFinder–open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes
Boyle EI, Weng S, Gollub J, Jin H, Botstein D, Cherry JM, Sherlock G.
Bioinformatics. 2004 Dec 12;20(18):3710-5. Epub 2004 Aug 5.
PubMed id: 15297299

Controlling the false discovery rate: a practical and powerful approach to multiple testing
Benjamini, Yoav; Hochberg, Yosef
Journal of the Royal Statistical Society. 1995, Series B (Methodological) 57 (1): 289–300.

Augmentation Procedures for Control of the Generalized Family-Wise Error Rate and Tail Probabilities for the Proportion of False Positives
van der Laan, Mark J.; Dudoit, Sandrine; and Pollard, Katherine S.
Statistical Applications in Genetics and Molecular Biology: Vol. 3 : Iss. 1, Article 15, 2004.

What's wrong with Bonferroni adjustments
Perneger, TV.
BMJ Publishing Group. 1998;316:1236.

Variable locus length in the human genome leads to ascertainment bias in functional inference for non-coding elements

Taher, L. and Ovcharenko, I. (2009), *Bioinformatics <http://bioinformatics.oxfordjournals.org/content/25/5/578>* Vol. : Iss. 5: 578–584.

Note: You can read more about **Hypergeometric Distribution** at Simple Interactive Statistical Analysis or Wolfram MathWorld. **Bonferroni Correction** is discussed in this Wolfram MathWorld article.

There are several list widgets (widgets from now on) available on the InterMine list analysis page, and they are configured in *Data and Widget Configuration*.

There are three categories of widgets:

table displays the counts from the list for the collection specified

graph displays a chart based on a dataset you specify

enrichment displays the p-values of objects that appear in your list

To add a widget to your mine:

1. add config to your `webconfig-model.xml` file
2. re-release your webapp
3. view widget in a list analysis page

Below are the details on how to configure each widget type.

Note: Please read the documentation carefully and check your config file for typos. Most attributes are case sensitive. When the webapp is released, the config is validated and any errors displayed in the home page.

Configuration

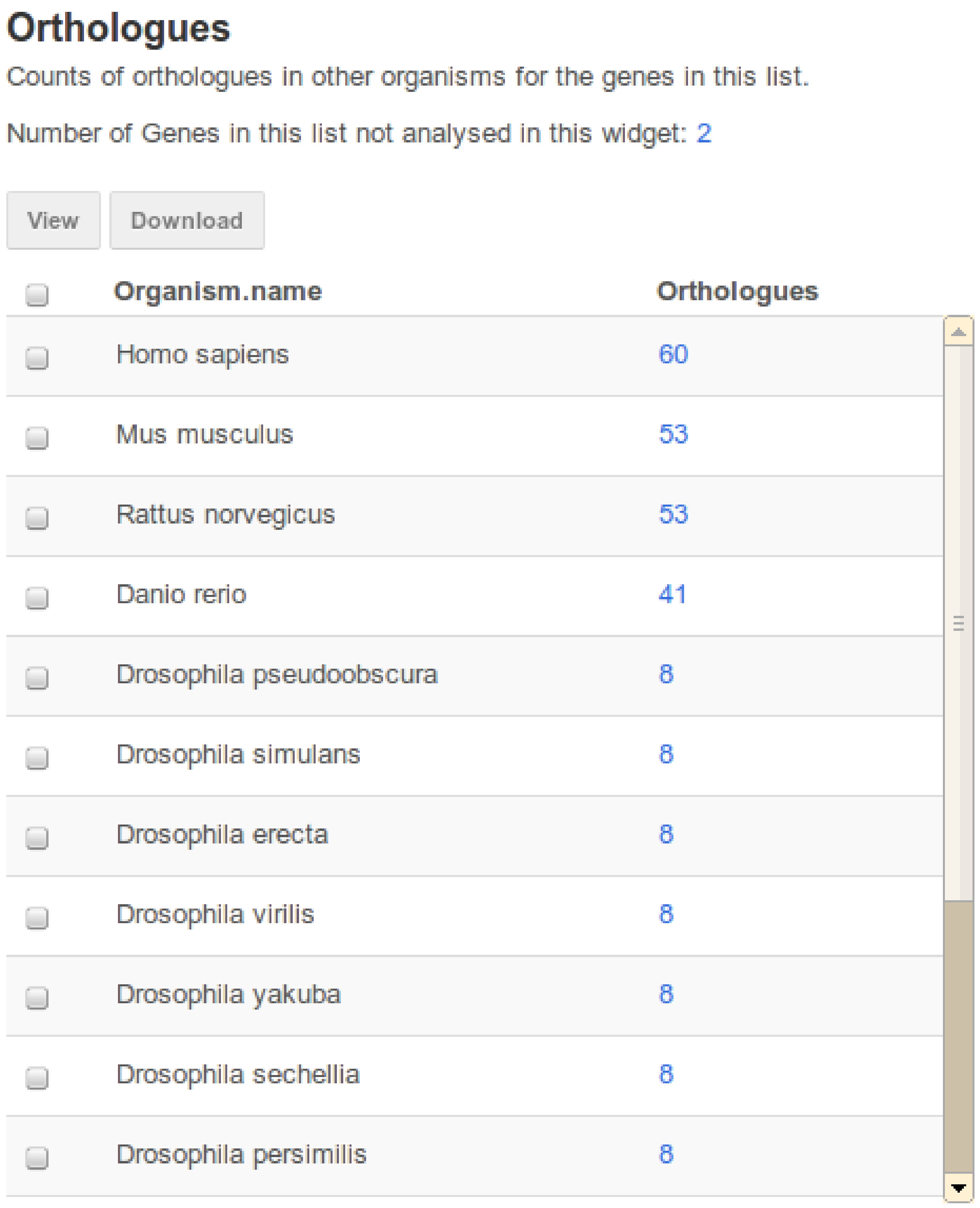
Table widgets

Table widgets display objects and the counts of related objects in your list.

An example table widget of Orthologues in FlyMine.

attribute	purpose	example
`id`	unique id used by javascript only. Spaces not allowed.	`unique_id`
`pathStrings`	which collection to use in the widget	`Gene.homologues[type=orthologue].homologue.orga`
`exportField`	which field from the objects in your list to export	`primaryIdentifier`
`typeClass`	types of lists that should display this widget. Use the simple class name	`Gene`

The following are optional attributes:



Orthologues

Counts of orthologues in other organisms for the genes in this list.

Number of Genes in this list not analysed in this widget: 2

Organism.name	Orthologues
Homo sapiens	60
Mus musculus	53
Rattus norvegicus	53
Danio rerio	41
Drosophila pseudoobscura	8
Drosophila simulans	8
Drosophila erecta	8
Drosophila virilis	8
Drosophila yakuba	8
Drosophila sechellia	8
Drosophila persimilis	8

attribute	purpose	example
`title`	appears at the top of the widget	`Orthologues`
`description`	description of the widget	`Counts of orthologues`
`displayFields`	which fields from the objects in the collection (in the above example, `Gene.proteins`) to display, eg. `primaryAccession`	`name`
`columnTitle`	heading for the "count" column	`Orthologues`
`externalLink`	link displayed next to first column, identifier will be appended to link	
`externalLinkLabel`	label for external link	
`views`	path fields display in the query running when the user clicks on the widget	`symbol`

Graph/Chart widgets

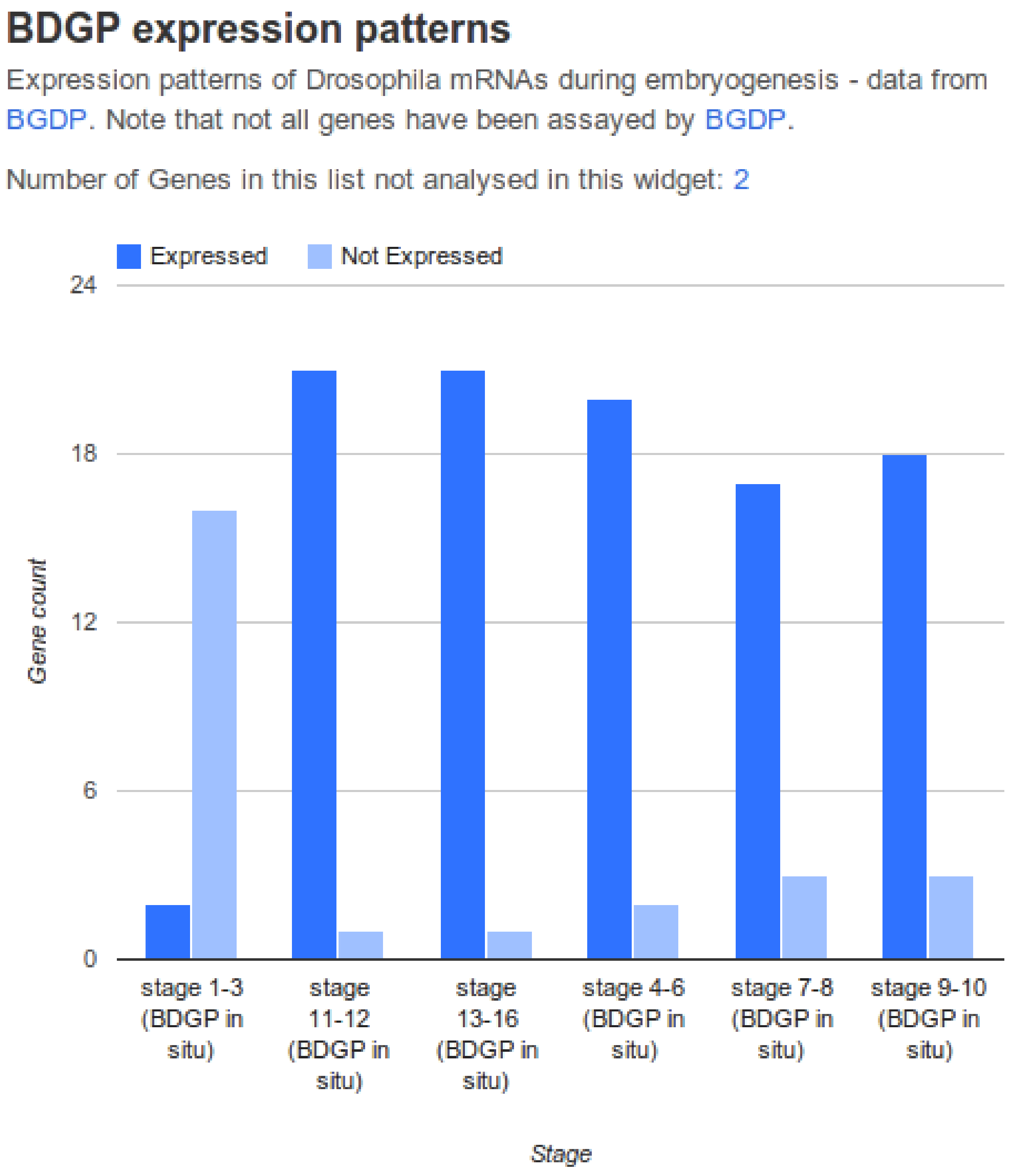
Graph widgets display datasets in graphical format.

attribute	purpose	example
id	unique id used by javascript only. Spaces not allowed.	`unique_id`
graphType	which type of chart to render	`ColumnChart`,``BarChart`` or `PieChart`
startClass	it's the root class for all the paths specified in the configuration [18].	`Gene`
typeClass	type of lists that should display this widget. Use the simple class name.	`Gene`
categoryPath	Must be attribute. We can specify the subclass using the syntax `path[subclass type]`	`mRNAExpressionResults.stageR`
seriesPath	the series path. This has to be an attribute. We can specify the subclass using the syntax `path[subclass type]`	`mRNAExpressionResults.expres`
seriesValues	the values of different series. Case sensitive. You can specify boolean values	`true,false` or `Up,Down`
seriesLabels	the labels displayed on the graphs to distinguish inside a category the different series	`Expressed,Not Expressed` or `Up,Down`
views	attributes paths displayed when the user clicks an area on the graph	`name,organism.name`

> **Warning:** You can specify **only one** class in `typeClass`. If you need another type, you have to define a new widget.

The following are optional attributes:

[18] All the paths set, will be built starting from that. Specify only the simple name (e.g. `Gene`). You can choose to set the bag type class or the root class associated to the category path.



BDGP expression patterns

Expression patterns of Drosophila mRNAs during embryogenesis - data from BGDP. Note that not all genes have been assayed by BGDP.

Number of Genes in this list not analysed in this widget: 2

Figure 1.24: An example chart widget of BDGP Expression Patterns in FlyMine.

attribute	purpose	example
`title`	appears at the top of the widget	`BDGP expression patterns`
`description`	description of the widget	`Expression patterns`
`domainLabel`	Label displayed on x-axis in the ColumnChart (on y-axis in the BarChart)	`Stage`
`rangeLabel`	Label displayed on y-axis in the ColumnChart (on x-axis in the a BarChart)	`Gene count`
`filterLabel`	label for filter form field	`Organism`
`filters`	the values for the filter, set in the dropdown [19].	`All,KEGG pathways,Reactome data`
`listPath`	the path used to build the bag constraint [20].	`FlyAtlasResult.material`
`constraints`	separated by comma, case sensitive, must be attributes, operator can be = or != [21]	`organism.name=[Organism]` [22]

Note: The graphs use Google Visualitation API.

Enrichment widgets

Enrichment widgets calculate p-values representing the probability annotation occurred by chance. See *List enrichment widgets statistics* for more information on how the p-value is calculated.

attribute	purpose	example
`id`	unique id used by JavaScript only. Spaces not allowed.	`unique_id`
`startClass`	Root class for all the paths specified in the configuration. Use simple name (e.g. Gene)	`Gene`
`startClassDisplay`	Field displayed when user clicks on the widget on 'Matches' column	`primaryIdentifier`
`typeClass`	Type of lists that should display this widget. Use the simple class name.	`Gene`
`enrich`	Field to be enriched, displayed in the widget in the firts column [23].	`goAnnotation.ontologyTerm.pare`
`views`	attributes paths displayed when the user clicks on *View results* button [6].	`symbol,organism.name`

Warning: You can specify **only one** class in `typeClass`. If you need another type, you have to define a new widget.

The following are optional attributes:

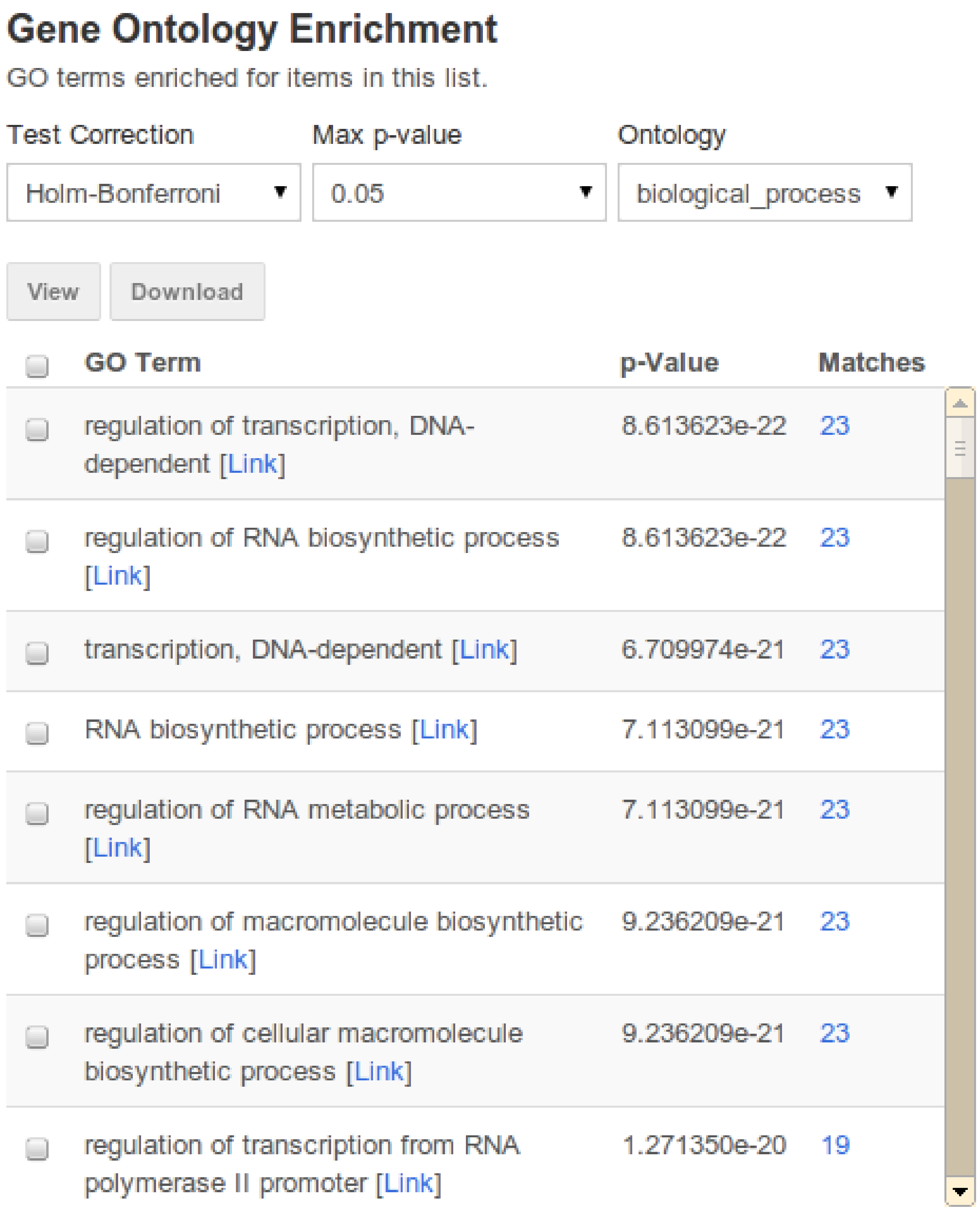
[19] We can use static values or a grammar to specify the values contained in the list. The default value in general is the first value set in the 'filters' attribute or the first value returned by the query. With static values, you can add 'All' meaning no filter applied.

[20] Optional if the `startClass` contains the bag type class.

[21] For the values we can use static values or the selected filter value using the syntax: `path constraint = [filter identifier]`.

[22] organism's name matching with the value selected in the filter with filterLabel 'Organism'

[23] You have to specify only one field. Specify the subclass using the syntax `path[subclass type]`.



Gene Ontology Enrichment

GO terms enriched for items in this list.

Test Correction: Holm-Bonferroni

Max p-value: 0.05

Ontology: biological_process

GO Term	p-Value	Matches
regulation of transcription, DNA-dependent [Link]	8.613623e-22	23
regulation of RNA biosynthetic process [Link]	8.613623e-22	23
transcription, DNA-dependent [Link]	6.709974e-21	23
RNA biosynthetic process [Link]	7.113099e-21	23
regulation of RNA metabolic process [Link]	7.113099e-21	23
regulation of macromolecule biosynthetic process [Link]	9.236209e-21	23
regulation of cellular macromolecule biosynthetic process [Link]	9.236209e-21	23
regulation of transcription from RNA polymerase II promoter [Link]	1.271350e-20	19

Figure 1.25: An example enrichment widget of Gene Ontology in FlyMine.

attribute	purpose	example
`title`	appears at the top of the widget	`Gene Ontology Enrichment`
`description`	description of the widget	`GO terms enriched.`
`label`	heading for the column	`GO Term`
`externalLink`	link displayed next to first column	`googie`
`filters`	extra filters to add to the display [24]	`organism.name=[list]`
`filterLabel`	label for filter form field	`Ontology`
`enrichIdentifier`	identifier for the row displayed, if not specified, enrich field used [25].	`goAnnotation.ontologyTerm.ide`
`constraints`	constraints separated by comma. The paths have to be attributes. The operator can be = or `!=` [26].	`organism.name=[list]`
`constraintsForView`	constraints separated by comma used for building the query executed when the user clicks on the widget on 'Matches' column	`results.expressed = true`
`correctionCoefficient`	set to org.intermine.bio.web.widget.GeneLenghtCorrectionCoefficient to normalize by gene length	

Examples

See other mines' config files for more examples, eg:

- FlyMine's webconfig-model.xml
- HumanMine's webconfig-model.xml

Background population

In the enrichement widgets, you can change the reference population. The reference population is specific for widget, list and user. If you are logged you can save your preference selecting the checkbox 'Save your preference'. The background population selected should include all items contained in the list.

Gene length correction coefficient

Depending on the type of experiment your data comes from, it is sometimes necessary to normalize by gene length in order to get the correct p-values. If your data comes from a genome-wide binding experiment such as ChIP-seq or DamID, binding intervals are more likely to be associated with longer genes than shorter ones, and you should therefore normalize by gene length. This is not the case for experiments such as gene expression studies, where gene length does not play a role in the likelihood that a particular set of genes will be overrepresented in the list. If you want normalize by gene length, add the attribute correctionCoefficient set to 'org.intermine.bio.web.widget.GeneLenghtCorrectionCoefficient'. The gene length correction coefficient is applicable only for lists containing genes with a length, so for a list of genes do not have a length the option is not shown. If a list contains some genes without a length these genes will be discarded.

Export Values

The exported file from enrichment widgets includes the enrichment identifier as the fourth column. It is contextual to the startClass attribute in the configuration. For example, an enrichment widget for publications would return the

[24] Use static values or a grammar to specify the values contained in the list. The default value in general is the first value set in the 'filters' attribute or the first value returned by the query. With static values, you can add 'All' meaning no filter applied.

[25] Specify only one. This has to be an attribute. Used in the results table. Specify the subclass using the syntax `path[subclass type]`.

[26] Case sensitive. For the values we can use: static values the selected filter value using the syntax: `path contraint = [filter identifier]` only the value contained in the list.

PubMedID field, where a GO enrichment widget would return the GO Term field.

Displaying widgets

JavaScript

Widget service Create a new Widgets instance pointing to a service:

```
var widgets = new intermine.widgets("http://beta.flymine.org/beta/service/");
```

Choose a widget Choose which widget(s) you want to load:

```
// Load all Widgets:
widgets.all('Gene', 'myList', '#all-widgets');
// Load a specific Chart Widget:
widgets.chart('flyfish', 'myList', '#widget-1');
// Load a specific Enrichment Widget:
widgets.enrichment('pathway_enrichment', 'myList', '#widget-2');
// Load a specific Table Widget:
widgets.table('interactions', 'myList', '#widget-3');
```

CSS

Note: Widgets are using Twitter Bootstrap CSS framework.

Embedding mine widgets on a custom page

Following is a documentation describing how to embed widgets not in a mine context.

1. Open up a document in your text editor.
2. Use the *InterMine JavaScript API Loader* that always gives you the latest version of the widgets. In the `<head>` element of the page, add the following line:

    ```
    <script src="http://cdn.intermine.org/api"></script>
    ```

3. Load the Widget Service:

    ```
    <script type="text/javascript">
        intermine.load('widgets', function() {
            var Widgets = new intermine.widgets('http://beta.flymine.org/beta/service/');
        });
    </script>
    ```

    `intermine.load` represents a block of code that loads the widgets by pointing them to a specific mine.

4. Use the widget web service to view which widgets are available on the mine, eg: *http://beta.flymine.org/beta/service/widgets/*
5. See which lists are available in the mine: *http://beta.flymine.org/beta/service/lists*
6. Add a widget (from the list in the previous step) to JavaScript. So within the `intermine.load` block, after creating the `Widgets` instance, do this:

```
// Load all Widgets:
Widgets.all('Gene', 'myList', '#all-widgets');
// Load a specific Chart Widget:
Widgets.chart('flyfish', 'myList', '#widget-1');
// Load a specific Enrichment Widget:
Widgets.enrichment('pathway_enrichment', 'myList', '#widget-2');
// Load a specific Table Widget:
Widgets.table('interactions', 'myList', '#widget-3');
```

Where the *first parameter*‘ passed is either type of object or name of widget to load. The *second* is the name of list (public list) to access and *third* is an element on the page where your widgets will appear. This element needs to obviously exist on the page first. A common one is a div that would look like this: `<div id="all-widgets"></div>`.

7. Add HTML, eg:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>test</title>
    <script src="http://cdn.intermine.org/api"></script>
    <script type="text/javascript">
        intermine.load('widgets', function() {
            var Widgets = new intermine.widgets('http://beta.flymine.org/beta/service/');
            // Load all Widgets:
            Widgets.all('Gene', 'myList', '#all-widgets');
        });
    </script>
</head>

<body>
    <!-- DIV goes here -->
    <div class="widget" id="all-widgets">
</body>
</html>
```

8. You will have noticed that the widgets either pickup a style (CSS) from your HTML page, or they appear unstyled. To style them, you can use a variant of Twitter Bootstrap.

1.8.2 Apps/C Grunt Build

Apps/C Usage

This document describes how to build JavaScript components using the Apps/C Grunt builder. It compiles CoffeeScript, JavaScript and Eco into CommonJS/1.1 Modules providing AMD/CommonJS/window external interface.

Example `Gruntfile`:

```
module.exports = (grunt) ->
    grunt.initConfig
        pkg: grunt.file.readJSON("package.json")

        apps_c:
            commonjs:
                src: [ 'src/**/*.{coffee,js,eco}' ]
                dest: 'build/app.js'
                options:
                    main: 'src/index.js'
                    name: 'MyApp'

    grunt.loadNpmTasks('grunt-apps-c')

    grunt.registerTask('default', [ 'apps_c' ])
```

You can now include the `build/app.js` file and, depending on your surrounding environment, you will be able to load it using RequireJS/AMD, CommonJS or straight from `window` under the `MyApp` key.

Config

The `options.main` property specifies which file will be considered the "main" one for your package. Somehow, the external world needs to know what to get when they call `require(package_name)`. If you do not specify this property the following actions are taken:

1. We try make use of the property `main` as specified in your app's `package.json` file. Failing that, we...
2. try to find the `index.[js|coffee]` file that is closest to the root of your sources.

The `options.name` overrides the name of the package in `package.json`. It specified the name of the exported package as in: `require(name)`. One can pass in an array of names, as alternatives, as well.

Eco Templates Are precompiled so when you require them, you need to only pass a `context` to them to get a string back.

CommonJS/1.1 Modules

The following template wraps your modules:

```
// filename
require.register('package/path.js', function(exports, require, module) {
  // ...
});
```

publication-search

Note: You can view the source files for this project in the intermine/intermine-apps-c repo.

This document will guide you through the process of writing a JavaScript client side app (running completely in a browser) using Bower and Grunt tools. This app will connect to an InterMine instance to run a query. The objective will be to fetch a list of publications for each *bio entity* found that is *like* our query.

The libraries we will be using:

1. Bower to fetch vendor dependencies such as JavaScript, CSS or Fonts.
2. canJS is a framework for client-side development handling routing, events etc.
3. CoffeeScript a language that compiles down to JavaScript and makes writing an app easier.
4. Foundation is a CSS framework of reusable UI components.
5. Grunt to build/transpile our source files.
6. jQuery is a DOM manipulation library (and more).

7. Mustache is a multi-platform templating language allowing us to embed dynamic objects in HTML.
8. Node JavaScript desktop software platform.
9. Stylus allows us to be more expressive and dynamic with CSS.
10. Lodash is a utility toolbelt making actions such as iterating over items easier.
11. imjs used to query InterMines from browser or Node. Saves you having to write raw HTTP requests.

Initialize Project

The first step will be to setup our directory structure.

build/ Will be the directory where our final app package will live. We will develop in languages like Stylus or CoffeeScript and need a way to package all these resources into one whole... directory. This is where all these files will live.

bower_components/ This directory will be automatically created and will contain libraries we have requested through the Bower system.

example/ Contains an example of our app in use.

src/ Source files that our code will consist of.

bower.json Will contain a listing of libraries we want to download using Bower.

package.json Lists libraries we will need to compile and build our app with.

Node.js platform Since our application is targeting JavaScript in the browser, it is pretty useful if we use JavaScript on our computer (desktop) too. Enter Node which allows us to execute JavaScript on our computers instead of just our browsers.

You can fetch binaries from the homepage or use your (hopefully Linux) packman.

Once Node is installed, edit the `package.json` file like so:

```
{
    "name": "publication-search",
    "version": "0.0.0",
    "devDependencies": {
        "bower": "~1.2.7",
        "grunt": "~0.4.1",

        "grunt-apps-c": "0.1.14",
        "grunt-contrib-concat": "~0.3.0",
        "grunt-contrib-stylus": "~0.9.0",

        "grunt-contrib-uglify": "~0.2.5",
        "grunt-contrib-cssmin": "~0.6.2"
    }
}
```

This file tells Node which libraries will be used to build our app. These are not client-side libraries, but server-side if you will.

The top bit of the `devDependencies` lists a bunch of Grunt and Bower related libraries.

In order to install all of these, execute the following:

```
$ npm install -d
```

Bower vendor dependencies Now we want to fetch libraries that our app, when running, will depend on.

Edit the `bower.json` file like so:

```
{
    "name": "publication-search",
    "version": "0.0.0",
    "dependencies": {
        "jquery": "2.0.3",
        "lodash": "2.4.1",
        "canjs": "2.0.4",
        "foundation": "5.0.2",
        "imjs": "3.2.1"
    }
}
```

The file has a bunch of key-value pairs.

name Name of our application in the Bower ecosystem, required.

version Version number in the Bower ecosystem, required.

dependencies Lists the actual libraries and their versions to fetch. You can populate this list by executing `$ bower install jquery --save` for example. That will download the latest version of the `jquery` component into the `bower_components/` directory. You can search for available components using `$ bower search jquery`. To actually trigger a search, execute `$ bower install`. The different libraries will be introduced as we code along.

Grunt building Grunt is used to munge files together and execute commands on them. Create a file called `Gruntfile.coffee`:

```
module.exports = (grunt) ->
    grunt.initConfig
        pkg: grunt.file.readJSON("package.json")

        apps_c:
            commonjs:
                src: [ 'src/**/*.{coffee,mustache}' ]
                dest: 'build/js/ps.js'
                options:
                    main: 'src/app.coffee'
                    name: 'ps'

        stylus:
            compile:
                src: [ 'src/styles/app.styl' ]
                dest: 'build/css/ps.css'

        concat:
            scripts:
                src: [
                    # Vendor dependencies.
                    'bower_components/jquery/jquery.js'
                    'bower_components/lodash/dist/lodash.js'
                    'bower_components/canjs/can.jquery.js'
                    'bower_components/canjs/can.map.setter.js'
                    'bower_components/imjs/js/im.js'
                    # Our app.
                    'build/js/ps.js'
                ]
                dest: 'build/js/ps.bundle.js'
                options:
                    separator: ';' # for minification purposes

            styles:
                src: [
                    'bower_components/foundation/css/normalize.css'
                    'bower_components/foundation/css/foundation.css'
                    # Our app.
                    'build/css/ps.css'
                ]
                dest: 'build/css/ps.bundle.css'

        uglify:
            scripts:
                files:
                    'build/js/ps.min.js': 'build/js/ps.js'
                    'build/js/ps.bundle.min.js': 'build/js/ps.bundle.js'

        cssmin:
            combine:
                files:
                    'build/css/ps.bundle.min.css': 'build/css/ps.bundle.css'
                    'build/css/ps.min.css': 'build/css/ps.css'

    grunt.loadNpmTasks('grunt-apps-c')
    grunt.loadNpmTasks('grunt-contrib-stylus')
    grunt.loadNpmTasks('grunt-contrib-concat')
    grunt.loadNpmTasks('grunt-contrib-uglify')
    grunt.loadNpmTasks('grunt-contrib-cssmin')

    grunt.registerTask('default', [
        'apps_c'
        'stylus'
        'concat'
    ])

    grunt.registerTask('minify', [
        'uglify'
        'cssmin'
    ])
```

This file is written in CoffeeScript and lists the tasks to run when we want to build our app. From the top:

apps_c This directive says that we want to take any CoffeeScript and Mustache files we find in `src/` and combine them into one JavaScript package.

stylus Take a Stylus file and turn it into CSS.

concat Take our vendor files (installed using Bower) and, together with our app, make them into a bundle. If someone else wants to use our app they need our app and its deps too, so this one file will do it for them. Do the same to CSS too.

uglify Minify our built JavaScript files. This makes them small, but unreadable so not great for debugging.

cssmin The same as *uglify* but for CSS

Then we have two calls to `grunt.registerTask` which bundle a bunch of tasks together. For example running `$ grunt minify` will run the `uglify` and `cssmin` tasks.

While developing it is quite useful to watch the source files and re-run the build task:

```
$ watch --color grunt
```

This will run the default Grunt task every 2s.

Source files

Example page One needs an access point where our app will get loaded with particular configuration. This is where the `example/index.html` comes in:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Publication Search</title>

    <link href="build/css/ps.bundle.css" media="all" rel="stylesheet" type="text/css" />
    <script src="build/js/ps.bundle.js"></script>
</head>
<body>
    <div id="app"></div>
    <script>
        // Once scripts have loaded.
        $(function() {
            // ...show the app.
            require('ps')({
                'el':   '#app',
                'mine': 'http://www.mousemine.org/mousemine'
            });
        });
    </script>
</body>
</html>
```

This file does not do anything else other then load our built CSS and JS files and starts our app once the page loads. In our example we are pointing to a `build` directory relative to the `example` directory. So let's make a symbolic link to the actual `build`:

```
$ ln -s ../build build/
```

Such links get preserved when version controlling using Git. We are linking to our bundled builds that contain vendor dependencies too.

Then we are waiting for the page to load and call our (future) app with some config.

The name `ps` is being configured in the `Gruntfile.coffee` file in the `apps-c` task.

As for the config:

el Selector where our app should be displayed.

mine Points to an InterMine.

The `require` call relates to CommonJS. It is one way of loading JavaScript modules. It avoids having to expose all of our functions and objects on the global (`window`) object and implements a way of relating between different files. For example, to load a module on the same *directory* level as me:

```
require './module'
```

App index We have asked to load an app in our `example/index.html` page, now we are going to write the backing code.

The `apps-c` task (in `Gruntfile.coffee`) contains the following two options:

name How do we call our app for CommonJS `require` call.

main Contains a path (an index) that will be called when we actually call the `require` function.

We have specified that our app index lives in `src/app.coffee` so let's create this file:

```
render   = require './modules/render'
query    = require './modules/query'
imjs     = require './modules/imjs'
state    = require './modules/state'

layout   = require './templates/layout'

components = [
    'alert'
    'search'
    'table'
]

module.exports = (opts) ->
    # Load the components.
    ( require "./components/#{name}" for name in components )

    # Setup the UI.
    $(opts.el).html render layout

    # Do we have mine set?
    return state.attr { 'type': 'warning', 'text': 'Mine is not set' } unless opts.mine

    # Setup the client.
    imjs.attr { 'client': new intermine.Service 'root': opts.mine }

    # Manually change the query to init the search?
    query(q) if q = opts.symbol
```

Each module (file) in our app needs to export some functionality. When we call `require` we will be getting this functionality.

Observable We are going to be using canJS which gives us objects that can be *observed*. What this means is that when their values change, others listening to these changes will be notified. When we want to change their value we call `attr` function on them. One such example is where we setup the client. We are passing an object which is set on *imjs* which is a canMap. Or the line below where we set a symbol on a *query* which is a canCompute. The advantage here is that whenever we set a new symbol on *query*, anyone else will be told it has changed and do something. This something means to trigger a search.

Components But first we are requireing some components into the memory. These are canComponent instances. They wrap some user interface functionality (think widget) and are tied to a DOM tag. Whenever this tag appears on the page, a component gets automatically created with the appropriate template and data. For now, let's just say these need to be loaded before we inject our first template into the page. An example of a tag:

```
<app-component></app-component>
```

We inject the said template, layout, on the line below. Layout will represent the HTML that is true for our app/page. It will have custom tags in it that automatically get rendered as components (as above).

Layout Let us take a look at the layout template then; in */src/templates/layout.mustache*:

```
<div class="row collapse">
    <div class="small-2 columns">
        <span class="prefix">Search:</span>
    </div>
    <div class="small-10 columns">
        <app-search></app-search>
    </div>
</div>

<div class="row collapse">
    <div class="small-12 columns">
        <app-alert></app-alert>
    </div>
</div>

<div class="row collapse">
    <div class="small-12 columns">
        <app-table></app-table>
    </div>
</div>
```

Our app will consist of 3 components:

app-search A component that will represent our input search field.

app-alert An alert message showing in what state the app is in.

app-table A table with results of our search.

Search component The search component will bind the *query* to our input field; in */src/components/search.coffee*:

```
query = require '../modules/query'

# Search form.
module.exports = can.Component.extend

    tag: 'app-search'

    template: require '../templates/search'

    scope: -> { 'query': { 'value': query } }

    events:
        'input keyup': (el, evt) ->
            if (evt.keyCode or evt.which) is 13
                query do el.val
```

To do so we need to require the *query* module. It is the same module we have seen in our app index. And then we are off using the standard canComponent notation. There is:

tag Which is the custom DOM tag/element for this component. Again, if this tag appears on the page, this component will spring to life.

template This is the template that will get injected into the *tag*.

scope Ah, the magic. You can either pass in an object of key-value pairs that will be accessible within our *template*. A more interesting approach is to return a function that returns said object. Doing so will make this component listen in on any changes in the object. In our example we are (using slightly convoluted notation) listening to changes to *query*, which is a canCompute.

events Makes this component listen to events in the template and then do something. The syntax is: <*selector*> <*event*>. In our example, whenever the user has pressed (and raised their finger) from a key on a keyboard, we call a function. This function checks that the key was *Enter* and updates the *query*.

Search template The search template just outputs the current value of the query:

```
<input type="text" placeholder="e.g. brca, gamma" value="{{ query.value }}" autofocus>
```

We are also giving this field the focus on the page so a user can just start typing.

Query module We have been talking about this *query* for a while, it is time to write its code; in */src/modules/query.coffee*:

```
pubs  = require './pubs'
imjs  = require './imjs'
state = require './state'

# The default search query.
query = can.compute ''

# Keep track of requests.
gid = 0

# Observe query changes to trigger a service search.
query.bind 'change', (ev, q) ->
    state.attr { 'type': 'info', 'text': 'Searching &hellip;' }
    id = ++gid

    imjs.search q, (err, res) ->
        # Too late?
        return if id < gid
        return state.attr { 'type': 'warning', 'Oops &hellip' } if err
        state.attr { 'type': 'success', 'text': "Found #{res.length} results" }
        pubs.replace res

module.exports = query
```

First we are requiring some other modules:

pubs Will represent our results collection/list.

imjs A module doing the actual search.

state Will be told what the state of the app is for alerts.

We initialize the query to be empty using ‘’. If a developer wants to pass an initial query, we have seen the relevant code in app index.

Then we have a function that listens in on our changes. Whenever query changes, this function is triggered. We use it to first say that we are starting a search. Then we actually call the *imjs* module to do the search. If all went fine, we inject the new results into the *pubs* module.

There are two things that could go wrong:

1. The search might not be succesfull (mine down, malformed query etc.)
2. The results may arrive too late when the user asks for another set of results before seeing the first set.

Both cases are handled.

State module Is a canMap that keeps track of the app state; it lives in */src/modules/state.coffee*:

```
module.exports = new can.Map
    'type': 'info'
    'text': 'Search is ready'
```

The map has two attributes, one for a type of state we are in *[info|success|warning]* and the other for the actual message.

IMJS module This module will do the actual search on the mine. It is called imjs since it is going to be using the imjs library behind the scenes. We will find it in */src/modules/imjs.coffee*:

```
query =
    'select': [
        'Publication.title'
        'Publication.year'
        'Publication.journal'
        'Publication.pubMedId'
        'Publication.authors.name'
        'Publication.bioEntities.symbol'
        'Publication.bioEntities.id'
    ]
    'orderBy': [
        { 'Publication.title': 'ASC' }
    ]
    'joins': [
        'Publication.authors'
    ]

module.exports = new can.Map

    # Needs to be initialized.
    client: null

    # Search publications by bio entity symbol.
    search: (symbol, cb) ->
        return cb 'Client is not setup' unless @client

        @client.query _.extend({}, query, {
            'where': [
                {
                    'path': 'Publication.bioEntities.symbol'
                    'op':   'CONTAINS'
                    'value': symbol
                }
            ]
        }), (err, q) ->
            return cb err if err
            # Run the query.
            q.tableRows (err, res) ->
                return cb err if err
                # Re-map to a useful format.
                remap = (rows) ->
                    type = null
                    _.extend _.zipObject(_.map rows, (row) ->
                        # Add our type.
                        type = row.class if row.column is 'Publication.bioEntities.id'
                        # Tuple of column - value.
                        [
                            row.column.split('.').pop()
                            if row.rows then _.map(row.rows, remap) else row.value
                        ]
                    ), { type }

                cb null, _.map res, remap
```

At the top we are defining the query that will be used to run the query. The format is that of an InterMine PathQuery. You can see imjs for syntax and more information. One can generate this syntax by visiting the mine in question, running a query in QueryBuilder and then choosing to export to JavaScript in the Results Table.

Our query will be looking for publications, fetching their bio entities (genes, alleles, proteins etc.) and authors. Authors is a separate collection mapped to a publication.

Then we are using the canMap syntax to define a *client* attribute and a *search* function. An object can have both attributes and functions defined.

We took care of initializing the *client* in app index. In that step, we were intiializing the imjs library to use a specific mine, MouseMine in our case.

The search function takes two parameters, a symbol and a callback. The first is the search symbol coming from *query* module, the second a function that will be called when we have errors or results. Hopefully the latter.

We are then using imjs syntax to extend our *query* with a constraint on a bio entity symbol, matching our symbol and returning *tableRows*.

The *remap* function is just formatting the results into a format that is useful to us. In our case we want to have the following data structure which is conducive to being traversed in a Mustache template:

```
[
    {
        "title": "Distinct negative regulatory mechanisms involved in the repression of human embryon
        "year": 1994,
        "journal": "J Biol Chem",
        "pubMedId": "7806539",
        "authors": [
            {
                "name": "Perez-Stable C",
                "type": null
            }
        ],
        "symbol": "Tg(Ggamma-T)15Cps",
        "id": 1678446,
        "type": "Transgene"
    }
]
```

We are extracting the type of the bio entity matched and creating a nested *authors* field.

Once we have the new data we are calling back using the *cb* function. It is customary to specify an error as the first argument into said function. Since all is well, we are passing a *null* value.

Publications list We still have one module to cover. This is the *pubs* we have refered to elsewhere; in */src/modules/pubs.coffee*:

```
module.exports = new can.List []
```

We are using the canList object to store an observable array of values. To be honest, we don’t need to use an observable object here, but you may want to if you are going to be changing values in the array rather than replacing the whole thing outright.

Alert component When doing our searches we have decided to keep track of the state of the application. Are we searching? Do we have errors? That sort of thing.

We already wrote a module, a canMap, to represent the data structure. Now we just need to write the canComponent for it.

```
state = require '../modules/state'

# An alert.
module.exports = can.Component.extend

    tag: 'app-alert'

    template: require '../templates/alert'

    scope: -> state
```

It does what it does. Which is to show up when *app-alert* appears and then display a template and observe when *state* changes.

Alert template Each component needs a template. the alert one will look like this:

```
<div class="alert-box {{ type }}">
    {{{ text }}}.
</div>
```

What we are saying here is to display a Foundation alert box with a custom type and a text. We use *{{{ }}}* to display the text which allows us to use HTML in the *text* string and have it unescaped.

Results table component Now that we are searching for and updating *pubs* with new data, we have to observe them in a canComponent and render them. In */src/components/table.coffee*:

```
pubs = require '../modules/pubs'

# Table of publication results.
module.exports = can.Component.extend

    tag: 'app-table'

    template: require '../templates/table'

    scope: -> { pubs }
```

This will make an array of publications available to us in a template under the *pubs* key.

Results table template As for the template that displays the results; in */src/templates/table.mustache*:

```
{{ #if pubs.length }}
<table>
    <thead>
        <tr>
            <th>Title</th>
            <th>Author(s)</th>
            <th>Journal</th>
            <th>Year</th>
            <th>Match</th>
        </tr>
    </thead>
    <tbody>
    {{ #pubs }}
        <tr>
            <td class="title">
                <a target="_{{ pubMedId }}" href="http://www.ncbi.nlm.nih.gov/pubmed/{{ pubMedId }}"
            </td>
            <td>
            {{ #authors }}
                <span class="author">{{ name }}</span>
            {{ /authors }}
            </td>
            <td>{{ journal }}</td>
            <td>{{ year }}</td>
            <td class="nowrap">
                <a target="_{{ id }}" href="http://www.mousemine.org/mousemine/report.do?id={{ id }}
                    {{ symbol }}
                </a>
                <span class="label">{{ type }}</span>
            </td>
        </tr>
    {{ /pubs }}
    </tbody>
</table>
{{ /if }}
```

Firstly we are checking if we actually have any results to speak of. If so we render a table <tr/> element for each publication.

We can see that *{{ #pubs }}* and *{{ #authors }}* both reresent a for loop.

Style We are going to wrap up by writing a stylesheet. For this we are going to use Stylus; in */src/styles/app.styl*:

```
@import 'nib'

body
    padding: 20px

table
    width: 100%

    td
        .author
            &:not(:last-child)
                &:after
                    content: ", "
                    display: inline-block
        .label
            padding: 0 4px
            line-height: 16px

        &.title
            width: 40%

        &.nowrap
            white-space: nowrap
```

Stylus allows us to write nested rules, such as when we want to select a table cell, *<td/>* in a *<table/>*.

At the top we can see a reference to nib. This will make any of our rules be generated with browser vendor prefixed, where appropriate and allows us to use shorthand notation for various oft repeated rules.

Fin

This concludes our application. Running a static web server to view the */example* folder we are presented with a page that displays our app. Typing a symbol into the input box and pressing *Enter* then launches a request against MouseMine and, if succesfull, shows us results.

Appendix

pomme.js What we have not covered is the case when we want to embed our app besides other apps on a page. If that were the case, all our CSS rules would start conflicting with other rules on the page. Not to speak of canComponents that may pop up in all kinds of places if we are using the same tags across different apps.

One way to deal with this issue is to make use of the pommejs library. What it does is create a sandbox (using an *<iframe/>*) which is isolated from anything else on the page. One would load an app inside one such sandbox and not have to worry about library collusion.

For example, we would create a pure pommejs *build* in Grunt; in *Gruntfile.coffee* add the following task:

```
copy:
    pomme:
        src: [ 'bower_components/pomme.js/build/app.bundle.js' ]
        dest: 'build/js/pomme.bundle.js'
        expand: yes
        flatten: yes

grunt.loadNpmTasks('grunt-contrib-copy')
```

This requires you to have the following task installed:

```
$ npm install grunt-contrib-copy
```

In order to download the library itself using Bower:

```
$ bower install pomme.js
```

Now we are copying a bundled version of pommejs into our build directory.

Do create this sandbox we are going to require pommejs instead; in */example/index.html*:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Publication Search</title>

    <script src="build/js/pomme.bundle.js"></script>
</head>
<body>
    <div id="app"></div>
    <script>
        // Once scripts have loaded.
        $(function() {
            var Pomme = require('pomme.js');
            var channel = new Pomme({
                'target': '#app',
                'template': function() {
                    return '<MY TEMPLATE HERE>'
                }
            });
        });
    </script>
</body>
</html>
```

In the section above we can see a placeholder for a template. In that place we need to return a string which will correspond to the html that needs to be executed within the sandbox. It should look something like this (but as a string!):

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Publication Search</title>

    <link href="build/css/ps.bundle.css" media="all" rel="stylesheet" type="text/css" />
    <script src="build/js/ps.bundle.js"></script>
</head>
<body>
    <div id="app"></div>
    <script>
        // Once scripts have loaded.
        $(function() {
            // ...show the app.
            require('ps')({
                'el':   '#app',
                'mine': 'http://www.mousemine.org/mousemine'
            });
        });
    </script>
</body>
</html>
```

So our example *index.html* has moved into a string and is being executed inside an iframe.

Refer to the pommejs documentation if you'd like to know how to open a two way communication channel between the parent page and the iframe window.

elastic-med

Note: @in-progress

Note: You can view the source files for this project in the intermine/intermine-apps-c repo.

This document will guide you through the process of writing a JavaScript client side app (running completely in a browser) using Bower and Grunt tools. The app will connect to an ElasticSearch (ES) instance to do *search*. ES wraps Apache Lucene and serves as a repository of indexed documents that one can search agains. If you prefer a short gist head over to *Apps/C Usage* instead.

The app will have the following functionality:

1. Work with *cancer* related publications from PubMed.
2. Ask user for an input text and get back a list of publications.
3. Click on any of the results to see a detailed view.
4. From the document detail search for publications *like* this one.
5. Autocomple and provide suggestions for user's input.

Among the important libraries we will be using:

1. Bower to fetch vendor dependencies such as JavaScript, CSS or Fonts.
2. canJS is a framework for client-side development handling routing, events etc.
3. CoffeeScript a language that compiles down to JavaScript and makes writing an app easier.
4. D3 is used to manipulate documents based on data.
5. ElasticSearch a search server with a RESTful web service peddling JSON documents.
6. Foundation is a CSS framework of reusable UI components.
7. Grunt to build/transpile our source files.
8. jQuery is a DOM manipulation library (and more).
9. Moment is a date library for parsing, manipulating and formatting dates.
10. Mustache is a multi-platform templating language allowing us to embed dynamic objects in HTML.
11. Node JavaScript desktop software platform.
12. Stylus allows us to be more expressive and dynamic with CSS.
13. Underscore is a utility toolbelt making actions such as iterating over items easier.

Warning: Some of the code block examples on this page feature line numbers. Please view the page in a widescreen mode.

Initialize Project

The first step will be to setup our directory structure.

build/ Will be the directory where our final app package will live. We will develop in languages like Stylus or CoffeeScript and need a way to package all these resources into one whole... directory. This is where all these files will live.

bower_components/ This directory will be automatically created and will contain libraries we have requested through the Bower system.

data/ Is a directory where we can keep data files that we will load to ES later.

example/ Contains an example of our app in use.

src/ Source files that our code will consist of.

bower.json Will contain a listing of libraries we want to download using Bower.

package.json Lists libraries we will need to compile and build our app.

Node.js platform Since our application is targeting JavaScript in the browser, it is pretty useful if we use JavaScript on our computer (desktop) too. Enter Node which allows us to execute JavaScript on our computers instead of just our browsers.

You can fetch binaries from the homepage or use your (hopefully Linux) packman.

Once Node is installed, edit the `package.json` file like so:

```
1   {
2       "name": "elastic-med",
3       "version": "0.0.0",
4       "devDependencies": {
5           "bower": "~1.2.7",
6           "grunt": "~0.4.1",
7
8           "grunt-apps-c": "0.1.10",
9           "grunt-contrib-concat": "~0.3.0",
10          "grunt-contrib-stylus": "~0.9.0",
11          "grunt-contrib-copy": "0.4.1",
12
13          "grunt-contrib-uglify": "~0.2.5",
14          "grunt-contrib-cssmin": "~0.6.2",
15
16          "elasticsearch": "1.0.1",
17          "coffee-script": "1.6.3",
18          "async": "0.2.9",
19          "lodash": "2.4.1"
20      }
21  }
```

This file tells Node which libraries will be used to build our app. These are not client-side libraries, but server-side if you will.

The top bit of the `devDependencies` lists a bunch of Grunt and Bower related libraries, the bottom one (*line 17 onward*) some libraries used to load ES with data.

In order to install all of these, execute the following:

```
$ npm install -d
```

Bower vendor dependencies Now we want to fetch libraries that our app, when running, will depend on.

Edit the `bower.json` file like so:

```
{
    "name": "elastic-med",
    "version": "0.0.0",
    "dependencies": {
        "jquery": "2.0.3",
        "lodash": "2.4.1",
        "canjs": "2.0.4",
        "elasticsearch": "http://cdn.intermine.org/js/elasticsearch.js/1.0.2/elasticsearch.jquery.js
```

```
            "moment": "2.4.0",
            "d3": "3.3.13",
            "colorbrewer": "1.0.0",
            "hint.css": "1.3.1",
            "foundation": "5.0.2",
            "font-awesome": "4.0.3",
            "simple-lru": "~0.0.2"
        }
}
```

The file has a bunch of key-value pairs.

name Name of our application in the Bower ecosystem, required.

version Version number in the Bower ecosystem, required.

dependencies Lists the actual libraries and their versions to fetch. You can populate this list by executing `$ bower install jquery --save` for example. That will download the latest version of the `jquery` component into the `bower_components/` directory. You can search for available components using `$ bower search jquery`. To actually trigger a search, execute `$ bower install`. The different libraries will be introduced as we code along.

Grunt building Grunt is used to munge files together and execute commands on them. Create a file called `Gruntfile.coffee`:

```
module.exports = (grunt) ->
    grunt.initConfig
        pkg: grunt.file.readJSON("package.json")

        apps_c:
            commonjs:
                src: [ 'src/**/*.{coffee,mustache}' ]
                dest: 'build/js/em.js'
                options:
                    main: 'src/app.coffee'
                    name: 'em'

        stylus:
            compile:
                src: [ 'src/styles/app.styl' ]
                dest: 'build/css/em.css'

        concat:
            scripts:
                src: [
                    # Vendor dependencies.
                    'bower_components/jquery/jquery.js'
                    'bower_components/lodash/dist/lodash.js'
                    'bower_components/canjs/can.jquery-2.js'
                    'bower_components/canjs/can.map.setter.js'
                    'bower_components/elasticsearch/index.js'
                    'bower_components/moment/moment.js'
                    'bower_components/colorbrewer/colorbrewer.js'
                    'bower_components/d3/d3.js'
                    'bower_components/simple-lru/index.js'
                    # Our app.
                    'build/js/em.js'
                ]
                dest: 'build/js/em.bundle.js'
                options:
                    separator: ';' # for minification purposes

            styles:
                src: [
                    'bower_components/foundation/css/normalize.css'
                    'bower_components/foundation/css/foundation.css'
                    'bower_components/hint.css/hint.css'
                    'bower_components/font-awesome/css/font-awesome.css'
                    'src/styles/fonts.css'
                    'build/css/em.css'
                ]
                dest: 'build/css/em.bundle.css'

        copy:
            fonts:
                src: [ 'bower_components/font-awesome/fonts/*' ]
                dest: 'build/fonts/'
                expand: yes
                flatten: yes

        uglify:
            scripts:
                files:
                    'build/js/em.min.js': 'build/js/em.js'
                    'build/js/em.bundle.min.js': 'build/js/em.bundle.js'

        cssmin:
            combine:
                files:
                    'build/css/em.bundle.min.css': 'build/css/em.bundle.css'
                    'build/css/em.min.css': 'build/css/em.css'

    grunt.loadNpmTasks('grunt-apps-c')
    grunt.loadNpmTasks('grunt-contrib-stylus')
    grunt.loadNpmTasks('grunt-contrib-concat')
    grunt.loadNpmTasks('grunt-contrib-copy')
    grunt.loadNpmTasks('grunt-contrib-uglify')
    grunt.loadNpmTasks('grunt-contrib-cssmin')

    grunt.registerTask('default', [
        'apps_c'
        'stylus'
        'concat'
        'copy'
    ])

    grunt.registerTask('minify', [
        'uglify'
        'cssmin'
    ])
```

This file is written in CoffeeScript and lists the tasks to run when we want to build our app. From the top:

apps_c This directive says that we want to take any CoffeeScript and Mustache files we find in `src/` and make them into one JavaScript package.

stylus Take a Stylus file and turn it into CSS.

concat Take our vendor files (installed using Bower) and, together with our app, make them into a bundle. If someone else wants to use our app they need our app and its deps too, so this one file will do it for them. Do the same to CSS too.

copy A task that copies fonts from FontAwesome into our build directory.

uglify Minify our built JavaScript files. This makes them small, but unreadable so not great for debugging.

cssmin The same as `uglify` but for CSS

Lines 76 and 83 have two calls to `grunt.registerTask` which bundle a bunch of tasks together. For example running `$ grunt minify` will run the `uglify` and `cssmin` tasks.

While developing it is quite useful to watch the source files and re-run the build task:

```
$ watch --color grunt
```

This will run the default Grunt task every 2s.

ElasticSearch

Start ElasticSearch ES will hold our index of publications. Fetch it and then unpack it somewhere.

To start it:

```
$ ./bin/elasticsearch
```

Check that it is up by visiting port `9200`. If you see a JSON message, it is up.

Load example publications To index some documents, use whichever client. I was using the JavaScript one and if you check the `data/` dir in `elastic-med` on GitHub you will be able to see one way that documents can be indexed. In that example:

```
$ ./node_modules/.bin/coffee ./data/index.coffee
```

That will index (after a few seconds) 1000 cancer publications found in `cancer.json`.

The `convert.coffee` file was used to convert source XML to JSON.

Check that documents got indexed by visiting the document URL in the browser:

You should get back a JSON document back provided you are using index `publications`, type `publication` and you have a document under the id `438`.

Source files

Example page One needs an access point where our app will get loaded with particular configuration. This is where the `example/index.html` comes in:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>ElasticMed</title>

    <link href="build/css/em.bundle.css" media="all" rel="stylesheet" type="text/css" />
    <script src="build/js/em.bundle.js"></script>
</head>
<body>
```

```
11         <div id="app"></div>
12         <script>
13             // Once scripts have loaded.
14             $(function() {
15                 // ...show the app.
16                 require('em')({
17                     'el': '#app',
18                     'service': 'http://newvegas:9200',
19                     'index':   'publications',
20                     'type':    'publication',
21                     'query':   'breast size exercise cancer'
22                 });
23             });
24         </script>
25     </body>
26     </html>
```

This file does not do anything else other then load our built CSS and JS files (*lines 7 and 9*) and starts our app. In our example we are pointing to a `build` directory relative to the `example` directory. So let's make a symbolic link to the actual `build`:

```
$ ln -s ../build build/
```

Such links get preserved when version controlling using Git. We are linking to our bundled builds that contain vendor dependencies too.

Then we are waiting for the page to load and call our (future) app with some config.

The name `em` is being configured in the `Gruntfile.coffee` file in the `apps-c` task.

As for the config:

el Selector where our app should be displayed.

service Points to the ES endpoint. By default it starts on port `9200`.

index Refers to the ES index we are using.

type Refers to the type of ES documents we are storing in our index.

query Is a default query we will want to show when our app loads.

The `require` call on *line 17* relates to CommonJS. It is one way of loading JavaScript modules. It avoids having to expose all of our functions and objects on the global (`window`) object and implements a way of relating between different files.

App index We have asked to load an app in our `example/index.html` page, now we are going to write the backing code.

The `apps-c` task (in `Gruntfile.coffee`) contains the following two options:

name How do we call our app for CommonJS `require` call.

main Contains a path (an index) that will be called when we actually call the `require` function.

We have specified that our app index lives in `src/app.coffee` so let's create this file:

```
module.exports = (opts) ->
    # Explode ejs options.
    { service, index, type } = opts

    # Init the ejs client.
```

```
6       ejs.attr { index, type, 'client': new $.es.Client({ 'hosts': service }) }
7
8       # Start routing.
9       new Routing opts.el
10      do can.route.ready
11
12      # Have we launched on the index?
13      if can.route.current('')
14          # Manually change the query to init the search.
15          query.attr 'current', opts.query or '' # '' is the default...
```

Each module (file) in our app needs to export some functionality. When we call `require` we will be getting this functionality.

We are going to be using canJS which consists of objects that can be *observed*. What this means is that when their values change, others listening to this changes will be notified. When we want to change their value we call `attr` function on them. One such example is on *line 7* where we change the value of `index`, `type` and `client` as passed in by the user from `example/index.html`.

$.es.Client Refers to ElasticSearch client in JavaScript which we have installed using Bower and munged in a bundle using Grunt as specified in `Gruntfile.coffee`.

Routing() Is a call to a future canControl component which will setup our routing. We need a way of change between an index page that does search and a detail page that shows a detail...

can.route.ready Actually tells canJS to start listening to changes in the browser address.

On *line 14* we see an example of checking whether we are looking at the index page when the app loads. If so we are changing a `current` attribute on a (futute) canMap component which will correspond to the query, meaning user query input. Our `example/index.html` page contains an example query to use in this case.

Router Now we need to write the actual router component. It will be a type of canControl and lives in the `src/app.coffee` file too. Since we do not want/need to export this functionality, it will be placed above the current `module.exports` call:

```
# Router switching between pages.
Routing = can.Control

    init: ->
        # Load the components.
        ( require "./components/#{name}" for name in components )

        # Setup the UI.
        layout = require './templates/layout'
        @element.html render layout, helpers

    # Index.
    route: ->
        template = require './templates/page/index'
        @render(template, {}, 'ElasticMed')

    # Document detail.
    'doc/:oid route': ({ oid }) ->
        fin = (doc) =>
            template = require './templates/page/detail'
            return @render(template, {}, 'ElasticMed') unless doc

            title = title.value if _.isObject title = doc.attr('title')
```

```
24              @render template, doc, "#{title} - ElasticMed"
25
26          # Find the document.
27          doc = null
28          # Is it in results?
29          if (docs = results.attr('docs')).length
30              docs.each (obj) ->
31                  # Found already?
32                  return if doc
33                  # Match on oid.
34                  doc = obj if obj.attr('oid') is oid
35
36          # Found in results cache.
37          return fin(doc) if doc
38
39          # Get the document from the index.
40          ejs.get oid, (err, doc) ->
41              # Trouble?
42              state.error err.message if err
43              # Finish with either a document or nothing
44              #  in which case (error will be shown).
45              fin doc
46
47      # Render a page. Update the page title.
48      render: (template, ctx, title) ->
49          @element.find('.content')
50          .html(render(template, ctx))
51          # Update title.
52          document.title = title
```

init We are loading some components that we are using in this app into the memory and then rendering our app layout. This layout will setup the structure for our whole app.

route Is a function that will be called when we are on the index page of the app. It renders the index page template.

doc/:oid route Matches when we are looking at a detail of a document/publication. So if someone manually types in the address `#!doc/438` or it changes as a result of user interaction, this function gets called. We are either retrieving the document from a results cache or we are explicitely calling for a document from ElasticSearch. Consider that when we search for documents, we get their content too so we do not need to fetch them again when looking at their *detail*. In contrast, someone could type in a random document address and we need to be ready for that. In either case we are calling the `fin` function on *line 19* to render the results.

render Serves as a helper we have created that injects a template into the DOM and updates the page title.

Pages templates When discussing the router we were talking about different page templates. Let us define them now.

In `src/templates/page/index.mustache`:

```
<p>ElasticSearch through a collection of cancer related publications from PubMed. Use <kbd>Tab</kbd>
<div class="page index">
    <app-search></app-search>
    <app-state></app-state>
    <app-results></app-results>
</div>
```

This is the index template with three custom tags corresponding to different components:

app-search the search form

app-state notification messages/titles

app-results the results when our search is successful

Now for the template that gets rendered on a detail page, in `src/templates/page/detail.mustache`:

```
<div class="page detail">
    <app-state></app-state>
    {{ #oid }}
    <div class="document detail">
        <app-document link-to-detail="false" show-keywords="true"></app-document>
    </div>
    <app-more></app-more>
    {{ /oid }}
<div>
```

We see that `app-state` is present, it will tell us when a doc is not found. If it is (we have a document `oid`) we show the rest of the page.

app-document Is the view of one document. We are passing extra parameters (options) into the context saying we don't want to link to the detail page (we are on detail page) but we want to show keywords (which will not be shown on the index results set).

app-more is a results set similar to `app-results` which corresponds to a component that will automatically search for and display documents that are similar like *this one*.

Application search template This template will be rendered for the `app-search` component defined on the index page. In `src/templates/search.mustache`:

```
<div class="row collapse">
    <div class="large-10 columns search">
        <div class="faux"></div>
        <input class="text" type="text" maxlength="100" placeholder="Query..." value="{{ query.curre
        {{ #if suggestions.list.length }}
        <ul class="f-dropdown suggestions" style="left:{{ suggestions.px }}px">
        {{ #suggestions.list }}
            <li {{ #active }}class="active"{{ /active }}>
                <a>{{ text }}</a>
            </li>
        {{ /suggestions.list }}
        </ul>
        {{ /if }}
    </div>
    <div class="large-2 columns">
        <a class="button secondary postfix">
            <span class="fa fa-search"></span> Search
        </a>
    </div>
</div>
{{ #if query.history.length }}
<div class="row collapse">
    <h4>History</h4>
    <ul class="breadcrumbs">
    {{ #query.history }}
        <li><a>{{ . }}</a></li>
    {{ /query.history }}
</div>
{{ /if }}
```

We are splitting the DOM into two parts. These parts have a `row` class on them representing the grid of the Foundation framework.

div.search The first part is split into two `columns`, one for the input field and the other for a button triggering search.

div.faux We will want to get caret position from the input field. To do that we are going to get all of the text from the input field up to the caret position and then copy it over to a div that has the same CSS styling as us, but is invisible. Then we are going to get the width of this element. `.faux` is this element.

input.text The place where input goes. We can see Mustache syntax here that outputs the value of the current query.

ul.suggestions Show up when a list of suggestions has some items. Represents suggestions for the current word, hence the need to get the caret position. If some suggestions are "active" (we hover on them etc.) then we toggle their CSS class.

ul.breadcrumbs A query history. Only shows up when it has items in it.

Application search component

1.8.3 Query Results

Query results can be configured in a number of ways, including:

export

See `export` for details on exporting options.

column headers

See *Using Class and Field Labels* to change column headers.

links

Only unique fields (class keys) are links in results pages. Add to *Class keys* to make the fields links on results pages. Instead of linking to an intermine report page, you can set the links to redirect to external page. See `redirects`

weird brackets

You may see the following in query results: *GO:0007480 [GOTerm]*. This happens when a column is a parent type but the individual result is a subclass. The subclass will by in brackets.

The initial Page Size

This can be configured on a table by table basis when the table is initialised:

```
$('#my-table').imWidget({
  type: 'table',
  url: 'www.flymine.org/query',
  query: {from: 'Gene', select: ['*'], where: {symbol: 'foo*'}},
  properties: { pageSize: 20 }
});
```

Icons

Two different icon style are supported, bootstrap *glyphicons* and *fontawesome*. These differ in the underlying technology they use, one using images (glyphicons) and the other SVG fonts (fontawesome). By using fonts fontawesome icons generally look a bit nicer, but they are not compatible with IE8. For this reason *glyphicons* are the default, and *fontawesome* must be selected explicitly:

```
intermine.setOptions({icons: 'fontawesome'}, '.Style');
```

To apply this setting in your current web-app, see *Setting Javascript Options*.

The initial state of Sub-Tables

Outer-Joined collections are rendered in subtables within a single cell. By default these are not immediately rendered, and just the number of rows are indicated. This means that even sections with very large sub-tables are rendered efficiently - in the worst case the sub-tables may contain thousands of rows, and so a table with even 10 main rows might present 10,000 sub-rows or more, which can significantly impact browser performance (an example of this would be a table that contained publications with an outer-joined selection of genes; genome publications can list every gene in an organism, and this scenario easily leads to very large sub-tables).

However, if you don't like the default behaviour and would prefer for the sub-tables to be open when the main table is rendered onto the page, this is simply altered, through the following configuration snippet:

```
intermine.setOptions({SubtableInitialState: 'open'})
```

If you would like to set this property on a table by table basis, then you must set the *SubtableInitialState* property to *open*, in the same manner as you would for pageSize.

```
$('#my-table').imWidget({
  type: 'table',
  url: 'www.flymine.org/query',
  query: {
    from: 'Gene',
    select: ['*', 'pathways.*'],
    where: {symbol: 'foo*'},
    joins: ['pathways']
  },
  properties: { SubtableInitialState: 'open' }
});
```

Cell Formatters

The cells in each table can be configured to display their information in custom manners. To do this a javascript function must be registered to handle certain types of cell, and configured to respond to certain paths.

Formatters are not enabled by default, as they may be unexpected, and in could cause unneccessary requests to the server. Fortunately they are easily enabled. There are four formatter included (but not enabled) by default:

- Location - formats a chromosome location as eg: "2L:123..456"
- Sequence - formats a DNA or Protein sequence in FASTA lines.
- Publication - formats a publication in a citable format with title, first author and year.
- Organism - formats an organism's name in italics, using the short-name format.

To enable these formatters register the formatted path (see below), eg:

```
intermine.scope('intermine.results.formatsets.genomic', {
  'Organism.name': true,
  'Organism.shortName': true
});
```

To enable all the default formatters, you can use the following snippet:

```
var keyPath, formatsets = intermine.results.formatsets.genomic;
for (keyPath in formatsets) {
  formatsets[keyPath] = true;
}
```

Such customisation javascript should be placed in a custom model-includes.js file.

The Formatting Function

The interface expected for a formatting function is:

```
(Backbone.Model intermineObject) -> String|HtmlElement
```

Where the Model instance represents an intermine object. Fields of the object can be retrieved through the standard `#get(String)` method. The return value will be inserted into the table using the `jQuery#html` function, so both html strings and HtmlElements can be accepted as return values.

This function is executed as a method on a intermine.results.table.Cell (which will be bound as `this`), supplying the following properties as part of its interface:

```
this.el :: HtmlElement - The cell element in the DOM.
this.$el :: jQuery - The cached jQuery selector for the cell element.
this.options :: Object - The arguments supplied when constructing the cell, this includes:
  options.query :: intermine.Query
```

The function may also support two optional parts of the formatter interface:

```
Formatter.replaces :: Array<String> - The list of fields of the class that this formatter replaces.
Formatter.merge :: (Backbone.Model, Backbone.Model) -> () - A function to merge information
  from different objects into a single model.
```

A typical pattern would be to check to see whether the object currently has all the information required to render it, and if not then make a request to retrieve the missing information. Any changes to the model will cause the cell to be re-rendered, thus a request that gets missing information and sets it onto the model will cause the function to be called again with the complete information.

For examples of implementations of this interface please see:

- https://github.com/intermine/im-tables/blob/dev/src/formatters/bio/core/organism.coffee
- https://github.com/intermine/im-tables/blob/dev/src/formatters/bio/core/chromosome-location.coffee

The Formatting Configuration

To register a function to respond to specific types of data, it must be referenced under the `intermine.results.formatters` namespace by the name of the class that it handles. For example this can be done with the `intermine.scope` function:

eg:

```
intermine.scope('intermine.results.formatters', {Exon: myExonFormatter});
```

A separate entry must be made under the 'intermine.results.formatsets.{modelname}' namespace to register which paths trigger cell formatting. For example to register a formatter for the 'Exon' class which only formats the 'symbol' field:

```
intermine.scope('intermine.results.formatsets.genomic', {'Exon.symbol': true});
```

In a similar way, we can disable any currently configured formatter by setting the value of this value to 'false':

```
intermine.scope('intermine.results.formatsets.genomic', {'Exon.symbol': false});
```

individual formatters can be configured to respond to different fields of an object. So you could have one formatter for *Gene.length* and another for *Gene.symbol*, if you are unable to achieve what you need with css alone. To do this, the value in the formatset should be the formatter itself, rather than a boolean value, eg:

```
intermine.scope('intermine.results.formatsets.genomic', {
  'Gene.symbol': geneSymbolFormatter,
  'Gene.length': geneLengthFormatter
});
```

Branding

Links to your site (or others) can be branded with icons. This is configurable by setting option as follows:

```
intermine.scope('intermine.options.ExternalLinkIcons',
  {"http://myhostname": "http://myhostname/my-branding.png"}
);
```

All links in table cells with the prefix *http://myhostname* will use the given image as a logo.

This requires that *intermine.options.IndicateOffHostLinks* is set to true.

1.8.4 InterMine JavaScript API Loader

See also:

GitHub repo for source code.

Note: If you are loading JavaScript libraries on a page you should use a loader (count of 1). Why not use ours?

Purpose

To simplify loading of CSS and JS libraries. The API Loader automatically works out the order the libraries should be loaded based on dependencies between them. It also skips libraries that already exist on a page or that pass a specific check.

How to use

Note: If you are passing a string or an Array as the first parameter into the library these are @deprecated but still working for backwards compatibility.

First you require the API Loader. You can for example use the following shorthand notation that always points to the latest version.

```
<script src="http://cdn.intermine.org/api"></script>
```

Now you can use the loader by passing in an object that looks for example like so:

```
intermine.load({
  'js': {
    'JSON': {
      'path': 'http://cdn.intermine.org/js/json3/3.2.2/json3.min.js'
    },
    'setImmediate': {
      'path': 'http://cdn.intermine.org/js/setImmediate/1.0.1/setImmediate.min.js'
    },
    'example': {
      'path': 'http://',
      'test': function() {
        return true;
      }
    },
    'async': {
      'path': 'http://cdn.intermine.org/js/async/0.2.6/async.min.js',
      'depends': ['setImmediate']
    },
    'jQuery': {
      'path': 'http://cdn.intermine.org/js/jquery/1.8.2/jquery.min.js',
      'depends': ['JSON']
    },
    '_': {
      'path': 'http://cdn.intermine.org/js/underscore.js/1.3.3/underscore-min.js',
      'depends': ['JSON']
    },
    'Backbone': {
      'path': 'http://cdn.intermine.org/js/backbone.js/0.9.2/backbone-min.js',
      'depends': ['jQuery', '_']
    }
  }
}, function(err) {
  // your libraries have loaded
});
```

The object works like so:

1. You pass in either a `js` or a `css` object based on whether you are requesting JavaScript or CSS libraries (or both).
2. The key inside the object, like `jQuery` then refers to your library. If this key is on a `window` object (as is the case with jQuery library), we won't load the library, it already exists.
3. If you do not like the previous check and want something more robust, pass a sync function under the `test` key. Return `true` if a library should NOT be loaded.
4. `path` represents the URL pointing to the library.
5. Use `depends` key passing an Array if a library depends on other libraries in your list. In the example you can see that `Backbone` depends on `jQuery` and _ (underscore.js). The appropriate loading order will be worked out from this.
6. Check the `err` var passed in the callback function (second parameter).

1.8.5 InterMine JavaScript Library

Please refer to this repo for more information.

1.8.6 Embedding examples

Please refer to this repo for various embedding examples using List Widgets, imjs and the like.

See also:

Report Displayers if you wish to embed a displayer on a report page only.

1.9 InterMine API Description

This section describes the public API definitions of parts of the InterMine system.

1.9.1 The PathQuery API

InterMine installations accept queries over their data in a custom format known as *Path-Queries*. This is a graph-based query format which inherits some of its semantics and terminology from SQL.

Paths

The core concept of *Path-Queries* is naturally enough the *Path*, examples of which are:

- `Gene`: A plain root
- `Gene.symbol`: A root and an attribute
- `Gene.chromosomeLocation`: A reference to a complex attribute (a reference)
- `Gene.organism.name`: A chain from a root to an attribute through one or more references.
- `Gene.pathways.identifier`: A path may potentially match multiple values - there may be several pathway identifiers that match this path for any given gene.
- `Protein.gene.homologues.homologue.alleles.alleleClass`: Paths may be of arbitrary length.

In the XML serialization of path-queries, all paths must be completely qualified. In the JSON format a prefix can be specified with the *from* or *root* property.

Queries

Queries associate paths with various parts of the query:

The View: Defining Output Columns

To define what is retrieved from the data-store, a view is defined. This is simply a list of paths; any information in the data-store graph that matches these paths and satisifies the constraints (see below) will be included in the results.

eg:

```
<query model="genomic" view="Organism.name Organism.taxonId"/>
```

```
{from: "Organism", select: ["name", "taxonId"]}
```

Joins: Handling null values

In any chain of references in a long path such as *Gene.sequence.residues* or *Gene.proteins.proteinDomains.name*, may be null. There are two behaviours supported for dealing with null references (ie. where a gene does not have any sequence attached, or it has not proteins, or those proteins have no protein domains).

- *INNER JOIN*: The default behaviour, this prevents the entire path from matching, so that if the query contains *Gene.symbol* and *Gene.proteins.name* and a gene in the data store has no proteins then that gene will not match at all, no data will be returned for the symbol of that gene - ie. it is a required feature of this query that all genes in the result set have at least one protein (this is a kind of implicit existential constraint).
- *OUTER JOIN*: Optional optional behaviour; this allows references in paths to be empty while permitting higher parts of the path to continue to match. So for example if the query contains *Gene.symbol* and *Gene.proteins.name* and a gene in the data store has no proteins then no protein data for that gene will be returned, but the gene will still match the query, and the symbol for that gene will be included in the retrieved results (this makes the proteins optional).

There are some consequences of using outer joins:

- Due to the optional nature of the outerjoined data, it is not permitted to sort on attributes in an outerjoined section
- Constraints (see below) cannot be combined in an *or* relationship across join boundaries. So one cannot ask for all genes which are either of a certain length or which have a certain pathway if there is an outer join on pathways.

eg:

```
<query model="genomic" view="Gene.symbol Gene.pathways.identifier">
  <join path="Gene.pathways" style="OUTER"/>
</query>
```

```
{from: "Gene", select: ["symbol", "pathways.identifier"], joins: ["pathways"]}
```

Constraints: Restricting matching values

By default all values of a given type match a query unless they are excluded by empty references on an inner joined path. To restrict the result set constraints can be used.

Constraints on attributes: The following are examples of constraints on attributes in the data store:

```
<constraint path="Gene.symbol" op="=" value="eve"/>
<constraint path="Gene.length" op="&gt;" value="12345"/>
<constraint path="Gene.homologues.homologue.organism.taxonId" op="!=" value="7227"/>
<constraint path="Gene.description" op="CONTAINS" value="some term"/>
```

The json format allows a couple of different mechanisms for describing constraints:

```
{
  select: ["Gene.symbol"],
  where: {
    "symbol": "eve",
    "length": {gt: 12345},
    "homologues.homologue.organism.taxonId": {"!=": 7227},
    "description": {contains: "some term"}
  }
}
```

or:

```
{
  select: ["Gene.symbol"],
  where: [
    {path: "symbol", op: "=", value: "eve"},
    {path: "length", op: ">", value: 12345},
    {path: "homologues.homologue.organism.taxonId", op: "!=", value: 7227},
    {path: "description", op: "CONTAINS", value: "some term"}
  ]
}
```

or

```
{
  select: ["Gene.symbol"],
  where: [
    [ "symbol", "=", "eve" ],
    [ "length", ">", 12345 ],
    [ "homologues.homologue.organism.taxonId", "!=", 7227 ],
    [ "description", "CONTAINS", "some term" ]
  ]
}
```

Multi-Value Constraints One can specifiy that a path resolve to a value matching one (or none) of a set of values:

```
<constraint path="Gene.symbol" op="ONE OF">
  <value>eve</value>
  <value>bib</value>
  <value>zen</value>
</constraint>
```

```
{
  select: ["Gene.proteins.name"],
  where: {
    symbol: ["eve", "bib", "zen"]
  }
}
```

A special sub-type of this kind of constraint is the range constraint:

```
<constraint path="Gene.chromosomeLocation" op="OVERLAPS">
  <value>X:12345..45678</value>
  <value>2L:12345..45678</value>
  <value>3R:12345</value>
</constraint>
```

```
{
  select: ["Gene.symbol"],
  where: {
    chromosomeLocation: {OVERLAPS: ["X:12345..45678", "2L:34567..78654", "3R:12345"]}
  }
}
```

Lookup Constraints Lookup constraints allow convenient constraints over multiple attributes of a value, or querying when you don't know the particular attribute you wish to constrain:

```
<constaint path="Gene" op="LOOKUP" value="eve"/>
```

```
{
  select: ["Gene.symbol"],
  where: [[ "Gene", "LOOKUP", "eve"]]
}
```

An extra disambiguating value can be supplied. Its meaning depends on context, so for example would limit genes to a particular organism:

```
<constaint path="Gene" op="LOOKUP" value="eve" extraValue="D. melanogaster"/>
```

```
{
  select: ["Gene.symbol"],
  where: [[ "Gene", "LOOKUP", "eve", "D. melanogaster"]]
}
```

List Constraints Nodes in the query graph can be constrained by membership in a stored list. This type of constraint is similar to multi-value constraints, in that we are looking at membership in a set, and also similar to lookup constraints in that we treat entities as subjects of the constraints, rather than values of any of the attributes of the entities. A simple example is selecting all the proteins for genes in a given list:

```
<constraint path="Protein.genes" op="IN" value="a given list"/>
<!-- Or to exclude those records -->
<constraint path="Protein.genes" op="NOT IN" value="a given list"/>
```

```
{
  select: ["Protein.*"],
  where: [["genes", "IN", "a given list"]]
}
```

The only relationships that may be asserted are "IN" and "NOT IN".

Loop Constraints Queries can require that two nodes in the query graph refer (or do not refer) to the same entity. This kind of constraint is termed a "Loop" constraint. An example of this is would be to request all the genes in the pathways a given gene is in, so long as they are (or are not) one of the orthologues of the gene in question.

A loop constraint is composed of two paths, and either = or *!=*.

```
<constraint path="Gene.homologues.homologue" op="=" value="Gene.pathways.genes"/>
<!-- or -->
<constraint path="Gene.homologues.homologue" op="!=" value="Gene.pathways.genes"/>
```

```
{
  select: ["Gene.homologues.homologue.*", "Gene.pathways.genes.*"],
  where: [
    ["Gene.symbol", "=", "x"],
    ["Gene.homologues.homologue", "=", "Gene.pathways.genes"]
  ]
}
```

Loop constraints must link paths that are not separated by *outer joins*.

Type Constraints Type constraints, in addition to limiting the returned results, have the side-effect of type-casting the references in their paths to the given type, enabling other paths to reference otherwise unrefereable fields.

```
<constraint path="Gene.overlappingFeatures" type="ChromosomeStructureVariation"/>
```

```
{
  from: "Gene",
  select: ["symbol", "overlappingFeatures.element1.primaryIdentifier"],
  where: {
    overlappingFeatures: "ChromosomeStructureVariation"
  }
}
```

Type constraints may not participate in the constraint logic, and as such never have a *code* associated with them.

Sort Order

The order of the results can be determined through the sort order:

```
<query model="genomic" view="Gene.symbol" sortOrder="Gene.length DESC Gene.name ASC"/>
```

```
{select: ["Gene.symbol"], sortOrder: [["length", "DESC"], ["name", "ASC"]]}
```

1.10 Support

1.10.1 Mailing list

Google Summer of Code list

To make a GSoC-related enquiry please email `gen-intermine-gsoc-public@lists.cam.ac.uk`, or to sign up for this list, visit https://lists.cam.ac.uk/mailman/listinfo/gen-intermine-gsoc-public

Developer list

Please join the InterMine developers mailing list `dev [at] lists [dot] intermine [dot] org` to receive updates and ask questions.

Join list

Archives of old messages are available here:

Message archive

1.10.2 Troubleshooting tips

This page describes what to do if you encounter problems when installing or maintaining InterMine. Please feel free to *Contact us* with any questions you may have.

Error messages

If you encounter a problem when running a task, try adding the verbose flag:

```
# add --stacktrace flag to get the complete error message
$ ./gradlew builddb --stacktrace
```

Logs

Data warehouse

When integrating data, usually the errors are in intermine.log in your mine's directory.

Webapp

When you see an error on the webapp or get a blank page and nothing appears in the webapp log from log4j, it is likely you will be able to find more information on what went wrong in the tomcat logs:

- tomcat/logs/catalina.out
- tomcat/logs/localhost.$DATE.logs

It will likely be the log that was modified last.

A good way of looking at the output to these logs in real time is to use the command:

```
$ tail -f tomcat/logs/$LOGNAME
```

If you reload the webapp you will see the error output directly on the screen.

IQL in logs

If you are having problems with a specific query, you can run it directly in the console. The query is listed in the log files in IQL (InterMine Query Language). To run the query directly, go to *$MINE/dmodel* and execute this command:

```
$ ./gradlew runIQLQuery -Pquery='some IQL'
```

Show all properties

Note that you can do this in a running web-app to check that it works by visiting the *HOST/PATH/showProperties.do* url when logged in as superuser.

Common Errors

Listed here are some common errors encountered in InterMine and some suggested fixes.

UnsupportedClassVersionError

```
java.lang.UnsupportedClassVersionError: org/intermine/task/FileName (Unsupported major.minor version
```

This means that your version of Java is too old, you need at least Java 8 to run InterMine.

can't open datasource

```
java.lang.RuntimeException: can't open datasource for {platform=PostgreSQL, datasource.dataSourceName
```

Errors of this type mean there was a problem accessing a database, in this example with *db.flatmodeunittest*. Either the database specified in the *mine.properties* file doesn't exist or the server/user/password details in the properties are incorrect.

FATAL: sorry, too many clients already

```
org.postgresql.util.PSQLException: Backend start-up failed: FATAL: sorry, too many clients already -
```

This occurs when the number of connections to a database exceeds the maximum configured in the postgres configuration. You need to increase the value of *max_connections* in the *postgresql.conf* file and restart postgres. Try 100 connections:

```
max_connections = 100
```

If you still experience this problem, restart Postgres.

OutOfMemoryError: Java heap space

```
java.lang.OutOfMemoryError: Java heap space
```

This means that a Java process hasn't been allocated enough memory to function correctly. You can increase the amount of memory by changing the *-Xmx* property in your *GRADLE_OPTS* environment variable. We recommend *8G* as a minimum, more is often needed during dataloading. Your *GRADLE_OPTS* variable should include the following:

```
$ echo $GRADLE_OPTS
$ -Xmx8G -Dorg.gradle.daemon=false
```

Can't find class name *ClassName*

```
Exception caught: java.lang.IllegalArgumentException: Unknown class name Protein in package org.inter
```

In this example a class named *Protein* could not be found in the data model, this will usually arise when running a parser and attempting to create an *Item* for a class that does not exist. Check your *SOURCE-NAME_additions.xml* files to see if the class is listed, only the additions files for sources lists on *project.xml* when *./gradlew builddb* was run will be included in the data model.

Can't find keys

```
Caused by: java.lang.IllegalArgumentException: Unable to find keys for source protfeatures_source in
```

It is expecting to find some keys to integrate data from that source. Do you have a keys file in the *protfeatures/src/main/resources*?

Classpath issues Classpath issues can generate various errors, eg a conflict caused by *wstx-asl-3.2.4.jar* when the XMLOutputFactory created its StreamWriter in PathQueryBinding:

```
XMLStreamWriter writer = factory.createXMLStreamWriter(sw);
```

Failed to parse the expression Tomcat 7 is less permissive than Tomcat 6, so you have might see this:

```
Caused by: org.apache.jasper.JasperException: /queryBuilderConstraint.jsp (line: 90, column: 14) "${
```

Add this to your Tomcat startup.sh script:

```
JAVA_OPTS="$JAVA_OPTS -Dorg.apache.el.parser.SKIP_IDENTIFIER_CHECK=true"
export JAVA_OPTS
```

See *Tomcat* for more details.

Session Error If you get a session error when you first start up your webapp, update your Tomcat configuration to handle different URLs. See *Tomcat*.

Client side errors

Assuming you are using Google Chrome as your browser press *Ctrl+Shift+I* to open a Debugger. In there click on the "Console" tab. If errors are present you should see them in red. If you want to inspect what kind of data are being sent/fetched, click on the Network Tab.

If you are using the List Widgets library `(>= 1.2.4)` then you can launch a "debug mode" on them. Simply wait for your page to load. Then append `#debug` at the end of the page URL. You should see buttons on individual widgets allowing you to see a debug trace of events happening.

Keyword Search fails There is no extra configuration required to get the search working. The search uses a Lucene index not the postgres database. The Lucene index is created at build-time, and it is the last source build as part of the tutorial.

The search should be very quick, but depending on the machine it's on, the initial search can be quite slow. On the first search, the index is unpacked from the database and loaded into memory which can take up to a minute.

If the search is just failing instantly, check your log files ($TOMCAT/logs). When the index is unpacked from the database, it writes to disk. There may be permissions or space issues.

See `/system-requirements/software/gradle/FAQs` for more error messages.

1.10.3 Contact us

To contact the InterMine Team:

email `info [at] intermine [dot] org`

twitter https://twitter.com/intermineorg

blog https://intermineorg.wordpress.com/

chat http://chat.intermine.org (Our public support channel on discord)

post

InterMine
Department of Genetics
Downing St
CAMBRIDGE CB2 3EH
United Kingdom

in person (please contact us first so we know to expect you) https://map.cam.ac.uk/Department+of+Genetics

1.11 About Us

InterMine is an open source data warehouse system for the integration and analysis of complex biological data, developed for the last 10 years by the Micklem Lab at the University of Cambridge. InterMine has been used for developing data warehousing solutions for a number of projects, including for storage and analysis of modENCODE data, and as a data mining platform for a number of major model organism databases as part of the InterMOD project.

InterMine has been developed with the support of the Wellcome Trust [067205], [082598], [090297], as well as support from the National Human Genome Research Institute [R01HG004834]. The Wellcome Trust also recently granted a further 5 years of funding for InterMine development, as well as development of HumanMine, a data warehouse of human genetic, genomic and proteomic data, ensuring continued development of InterMine as a framework.

The publicly available InterMine instances include:

- FlyMine - a data warehouse of integrated fruit fly genetic, genomic and proteomic data
- YeastMine - an integrated data warehouse of yeast genomic data, developed by SGD
- RatMine - an integrated data warehouse of rat genomic data, developed by RGD
- MouseMine - an integrated data warehouse of mouse genomic data, developed by MGI
- TargetMine - a data warehouse for candidate gene prioritisation and drug target discovery, developed at NIBIO, Japan
- Zebrafishmine - an integrated data warehouse of zebrafish genomic data, developed by ZFIN
- Thalemine - a data warehouse for Arabidopsis thaliana Col-0 for the ARAPORT project
- PhytoMine - an integrated data warehouse of over 50 plant genomes from Phytozome.

See the InterMine registry for the full list of InterMine instances.

More information:

1.11.1 Contact us

To contact the InterMine Team:

email `info [at] intermine [dot] org`

twitter https://twitter.com/intermineorg

blog https://intermineorg.wordpress.com/

chat http://chat.intermine.org (Our public support channel on discord)

post

InterMine
Department of Genetics
Downing St
CAMBRIDGE CB2 3EH
United Kingdom

in person (please contact us first so we know to expect you) https://map.cam.ac.uk/Department+of+Genetics

1.11.2 How to cite us

If you use the InterMine framework in your research, we would appreciate it if you cite the following publication:

- InterMine: extensive web services for modern biology. Kalderimis A, Lyne R, Butano D, Contrino S, Lyne M, Heimbach J, Hu F, Smith R, Stěpán R, Sullivan J, Micklem G. Nucleic Acids Res. 2014 Jul; 42 (Web Server issue): W468-72
- InterMine: a flexible data warehouse system for the integration and analysis of heterogeneous biological data. Smith RN, Aleksic J, Butano D, Carr A, Contrino S, Hu F, Lyne M, Lyne R, Kalderimis A, Rutherford K, Stepan R, Sullivan J, Wakeling M, Watkins X, Micklem G. Bioinformatics (2012) 28 (23): 3163-3165.

See the InterMine zotero group for the full list of InterMine publications.

1.11.3 Legal

All InterMine code is freely available under the open source LGPL license.

1.11.4 Privacy Policy

Privacy policy

This privacy policy sets out how InterMine uses and protects any information that you give when you use InterMine websites.

InterMine is committed to ensuring that your privacy is protected. Should we ask you to provide certain information by which you can be identified when using InterMine websites, then you can be assured that it will only be used in accordance with this privacy statement.

What we collect

Tracking We log the IP address of your browser to track usage statistics and to identify operational problems. This information is not used to identify individuals or organizations, and is never shared with third parties.

Cookies are used to provide persistence across browsing sessions. These may persist after you exit your browser, but they are never used for either identification or tracking purposes.

Login Details If you choose to create an account to save your data, we save your username and password information. This information is not used to identify individuals or organizations, and is never shared with third parties.

What we do with the information we gather

We require this information to understand your needs and provide you with a better service, and in particular for the following reasons:

- Tracking **usage statistics**.
- Identifying **operational problems**.
- Allowing you to **log in** and save your data.

The information we collect is not used to identify individuals or organizations, and is never shared with third parties.

Security

We are committed to ensuring that your information is secure. In order to prevent unauthorised access or disclosure, we have put in place suitable procedures to safeguard and secure the information we collect online.

How we use cookies

A cookie is a small file which asks permission to be placed on your computer's hard drive. Once you agree, the file is added and the cookie helps analyse web traffic or lets you know when you visit a particular site. Cookies allow web applications to respond to you as an individual. The web application can tailor its operations to your needs, likes and dislikes by gathering and remembering information about your preferences.

We use traffic log cookies to identify which pages are being used. This helps us analyse usage data and identify operational problems. We only use this information for statistical analysis purposes and then the data is removed from the system. Overall, cookies help us provide you with a better website, by enabling us to monitor which pages you find useful and which you do not, and saving your lists and queries across browsing sessions. A cookie in no way gives us access to your computer or any information about you, other than the data you choose to share with us.

You can choose to accept or decline cookies. Most web browsers automatically accept cookies, but you can usually modify your browser setting to decline cookies if you prefer. This may prevent you from taking full advantage of the website.

Links to other websites

Our website may contain links to other websites of interest. However, once you have used these links to leave our site, you should note that we do not have any control over that other website. Therefore, we cannot be responsible for the protection and privacy of any information which you provide whilst visiting such sites and such sites are not governed by this privacy statement. You should exercise caution and look at the privacy statement applicable to the website in question.

Controlling your personal information

We will not sell, distribute or lease your personal information to third parties. We will only use your personal information to send you information if you specifically sign up to our mailing lists, and you can opt out of receiving these at any time.

1.12 InterMine Video Tutorial Collection

Welcome to InterMine's online video tutorial collection. Here you can find out how to work with InterMine databases. You'll also find a PDF version of each tutorial so that you can work through the examples off-line.

1.12.1 Getting Started

If you're new to InterMine it's probably best to see the 'FlyMine Lightning Overview' tutorial first.

- **Getting Started with InterMine**

  A quick introduction to InterMine's web interface using the FlyMine database.

1.12.2 Lists and Template Searches

Lists: InterMine's List creation tool helps you upload a List of identifiers - most commonly Genes, Proteins or SNPs. See how to upload a list of Gene.

Templates: To help with data analysis, InterMine includes a library of Template searches - predefined searches designed to perform a particular task. Each one has a description and a form to fill in.

- **List Upload**

  This tutorial shows you how to upload a List of Gene identifiers.

- **Using Template Searches**

  This tutorials shows you to analyse data with InterMine's predefined Template searches.

See http://intermine.org/tutorials/ for a list of all available videos.

CHAPTER 2

Indices

- *genindex*

This guide in a PDF format

- *search*

Index