1 Contents
  1.1 System Requirements ......................................................... 3
  1.2 Get the Software .............................................................. 19
  1.3 Get started ................................................................. 20
  1.4 Data Model ................................................................. 50
  1.5 Database ................................................................. 58
  1.6 Guide to Customising your Web Application ......................... 121
  1.7 Web Services ............................................................ 208
  1.8 Embedding InterMine components ..................................... 209
  1.9 InterMine API Description ............................................... 258
  1.10 Support ................................................................. 262
  1.11 About Us ............................................................... 270
  1.12 InterMine Video Tutorial Collection ................................. 275

2 Indices ................................................................. 277
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.

This guide in a PDF format
System Requirements

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 = 1539238252
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.

Software

InterMine makes use of a variety of freely available software packages. All of the software listed below is required.
You should try to use the latest versions of the above software.

**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:

### Ant

#### Installation Notes

Several people have had problems with Ant installations set up by Linux package managers. We recommend installing Ant manually.

[Ant’s manual](#) has detailed instructions on how to install Ant.

Make sure you don’t have any ant-related jars on your classpath already, or else the InterMine build will fail.

#### InterMine and Ant

The InterMine project uses Ant for compiling code, building data models and databases and for running the data loading/integration process. You shouldn’t ever have to write new Ant XML, but you will run lots of Ant targets we have already defined. These will usually compile some code, create databases or run particular Java code.

An example use is to create the tables and indexes for a new Mine in an empty PostgreSQL database. In the `<mine>/dbmodel` directory we would do:

```bash
$ ant build-db
```

Whenever you use `ant` you can add a `--verbose` flag to get verbose output. We recommend doing this as it will give you detailed stack traces if a command fails:

```bash
$ ant -v build-db
```

You can pass parameters from the command line using `-Dparameter=value`. For example, when integrating data you can pass in a particular source name:

```bash
$ ant -v -Dsource=malaria-gff
```

Ant and the InterMine build system will ensure that all prerequisite code is compiled, the model is created and will create all tables and indexes based on the model.
Common Ant targets

<table>
<thead>
<tr>
<th>Target</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean</td>
<td>clean this project and all dependencies</td>
</tr>
<tr>
<td>clean-all</td>
<td></td>
</tr>
<tr>
<td>compile</td>
<td>produce jar/war</td>
</tr>
<tr>
<td>jar</td>
<td>create generated code</td>
</tr>
<tr>
<td>generate</td>
<td>do the action for this project, if any</td>
</tr>
<tr>
<td>action</td>
<td>build javadoc to build/javadoc</td>
</tr>
</tbody>
</table>

Tests

<table>
<thead>
<tr>
<th>Target</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>will produce a jar and run tests</td>
</tr>
<tr>
<td>test</td>
<td></td>
</tr>
<tr>
<td>test-all</td>
<td>test all dependencies as well as this project</td>
</tr>
</tbody>
</table>

To run a single test class:

```
$ ant -Dtest.includes=org/intermine/ClassToTest.class
```

DBmodel

<table>
<thead>
<tr>
<th>Target</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>build-db</td>
<td>build the objectstore</td>
</tr>
<tr>
<td>run-iql-query</td>
<td>run an IQL query directly in the console</td>
</tr>
</tbody>
</table>

Webapp

<table>
<thead>
<tr>
<th>Target</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean</td>
<td>removes temporary directories</td>
</tr>
<tr>
<td>default</td>
<td>forces rebuild of .war file</td>
</tr>
<tr>
<td>release-webapp</td>
<td>deploy to tomcat</td>
</tr>
<tr>
<td>remove-webapp</td>
<td>remove from tomcat</td>
</tr>
<tr>
<td>build-db-userprofile</td>
<td>initialise/clear the userprofile database (loads default templates too)</td>
</tr>
<tr>
<td>load-default-templates</td>
<td>load default-template-queries.xml to the superuser account</td>
</tr>
<tr>
<td>write-userprofile-xml</td>
<td>exports user profile database xml file to build/userprofile.xml</td>
</tr>
<tr>
<td>read-userprofile-xml</td>
<td>imports user profile database from build/userprofile.xml</td>
</tr>
<tr>
<td>drop-precomputed-tables</td>
<td></td>
</tr>
<tr>
<td>precompute-queries</td>
<td></td>
</tr>
<tr>
<td>precompute-queries-test</td>
<td></td>
</tr>
<tr>
<td>precompute-templates</td>
<td></td>
</tr>
</tbody>
</table>

Test-model

<table>
<thead>
<tr>
<th>Target</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert-data-unittest</td>
<td>insert test data into database</td>
</tr>
</tbody>
</table>

Git

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

1. install the Git command line tool
   
   ```
   $ sudo apt-get install git-core
   ```

2. configure your user and email

1.1. System Requirements
$ git config --global user.name "Name Surname"
$ git config --global user.email "your.email@gmail.com"

3. Our code is available on GitHub
   (a) create your account in github
   (b) login and go in ‘Account Settings’ section to configure the SSH keys

Java

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

```
$ export ANT_OPTS="-server -XX:MaxPermSize=256M -Xmx1700m -XX:+UseParallelGC -Xms1700m -XX:SoftRefLRUPolicyMSPerMB=1 -XX:MaxHeapFreeRatio=99"
```

Run `~/.bashrc` 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.

Increase the MaxPermSize setting if you get this error `java.lang.OutOfMemoryError: PermGen space`

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:
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 libxml-parser-perl
```

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:

#### 1.1. System Requirements
• 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
InterMine Documentation, Release

• Digest::MD5

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

<table>
<thead>
<tr>
<th>listen_addresses</th>
<th>port</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>5432</td>
</tr>
</tbody>
</table>

Install Bioseg  If you are installing a version of PostgreSQL below 9.2 you will need to install Bioseg to allow Range Queries. See the link for instructions. On PostgreSQL 9.2 and above the necessary functionality is built-in.

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

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.

1.1. System Requirements
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='C';
\c template1
update pg_database set datistemplate = TRUE where datname = 'template1';
\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

```
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](http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server) for more information.
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Suggested value (build)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared_buffers</td>
<td>10-25% of RAM</td>
</tr>
<tr>
<td>temp_buffers</td>
<td>around 80MB</td>
</tr>
<tr>
<td>work_mem</td>
<td>around 500MB but &lt; 10% of RAM</td>
</tr>
<tr>
<td>maintenance_work_mem</td>
<td>5% of RAM but &lt; 20% of RAM</td>
</tr>
<tr>
<td>default_statistics_target</td>
<td>around 250</td>
</tr>
<tr>
<td>random_page_cost</td>
<td>around 2.0-2.5</td>
</tr>
<tr>
<td>effective_cache_size</td>
<td>50% of RAM</td>
</tr>
<tr>
<td>synchronous_commit</td>
<td>off</td>
</tr>
<tr>
<td>geqo_threshold</td>
<td>14</td>
</tr>
<tr>
<td>fromCollapse_limit</td>
<td>14</td>
</tr>
<tr>
<td>joinCollapse_limit</td>
<td>14</td>
</tr>
<tr>
<td>max_locks_per_transaction</td>
<td>640</td>
</tr>
<tr>
<td>max_pred_locks_per_transaction</td>
<td>640</td>
</tr>
<tr>
<td>checkpoint_segments</td>
<td>128</td>
</tr>
<tr>
<td>checkpoint_timeout</td>
<td>10min</td>
</tr>
<tr>
<td>checkpoint_completion_target</td>
<td>0.9</td>
</tr>
</tbody>
</table>

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 templatel with template = template0 encoding = 'SQL_ASCII' LC_COLLATE='C' LC_CTYPE='C';
**CREATE DATABASE**

**template0=#** update pg_database set datistemplate = TRUE where datname = 'templatel';
**UPDATE 1**

```
templatel=# \c templatel
You are now connected to database "templatel" as user "postgres".
```

**template1=#** update pg_database set datallowconn = FALSE where datname = 'template0';
**UPDATE 1**

### Range Queries

**Important:** Starting with InterMine 1.4, bioseg is no longer required if you are using a PostgreSQL version higher than 9.1.
InterMine now can perform range queries on location table taking advantage of PostgreSQL built in `int4range` type.

The `int4range` type requires PostgreSQL 9.2 and queries perform best with 9.3.

If range types are not available (PostgreSQL 9.1) then queries will revert to bioseg if available or vanilla queries if not. New installations of PostgreSQL may still need to install gist for the indexes to work.

**Indexing**  
Performance is improved by creating an index of `int4range(start, end)` on the location table.

This is achieved in the post-process phase of the build of an InterMine (see Tutorial).

Please use the `create-location-range-index` post-process, which should replace `create-bioseg-location-index` in the project.xml file in your YOUR_MINE directory.

The `create-overlap-view` task will also detect whether built-in ranges are available and use them instead of bioseg for the overlappingfeatures view.

**PostgreSQL 9.1**  
For PostgreSQL 9.1 bioseg is still required. Here is how to install it.

**Prerequisites**

### Contrib software

<table>
<thead>
<tr>
<th></th>
<th>debian</th>
<th>fedora</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>apt-get install postgresql-contrib-9.1</code></td>
<td><code>yum install postgresql-contrib</code></td>
</tr>
</tbody>
</table>

### Header files

<table>
<thead>
<tr>
<th></th>
<th>debian</th>
<th>fedora</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>apt-get install postgresql-server-dev-9.1</code></td>
<td><code>yum install postgresql-devel</code></td>
</tr>
</tbody>
</table>

### pg_config

<table>
<thead>
<tr>
<th></th>
<th>debian</th>
<th>fedora</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>apt-get install libpq-dev</code></td>
<td></td>
</tr>
</tbody>
</table>

**Installation**

2. Untar the file, change to the created directory and run these commands
   
   ```bash
   $ make USE_PGXS=t clean
   $ make USE_PGXS=t
   $ make USE_PGXS=t install
   ```

**Create bioseg Type**  
You need to create the bioseg type in each database that is going to use it. If you create it in the `template1`, then all newly-created databases will have the bioseg type.

**Warning:** DO NOT install bioseg to the `template0` or `postgres` databases - they should never be altered.

Change directory to the postgres contrib directory

<table>
<thead>
<tr>
<th></th>
<th>debian</th>
<th>fedora</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>/usr/share/postgresql/9.1/contrib</code></td>
<td><code>/usr/share/pgsql/contrib</code></td>
</tr>
</tbody>
</table>

For each database, type:
Gist  We also need to create the default gist operators for Bioseg too, in order to have normal types in multi-column indexes.

**PostgreSQL 8.x users**  For each database, type:

```
# in the contrib directory
$ psql (database) <btree_gist.sql
```

**PostgreSQL 9.x users**  See [http://www.postgresql.org/docs/9.1/static/btree-gist.html](http://www.postgresql.org/docs/9.1/static/btree-gist.html). Run the command in the template1 database:

```
$ CREATE EXTENSION btree_gist;
```

**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*:

```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](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:

---

1.1. System Requirements
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:

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

Without this, permalinks may break.

startup.sh Add this property in startup.sh:

```bash
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:

```bash
# 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:

```bash
# 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 manager 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
```


**Installing Postgres**

See [http://www.postgresql.org/download/macosx](http://www.postgresql.org/download/macosx). We’ve had the most success with MacPorts.

**Eclipse**

The InterMine checkout includes configuration to open the various code modules as Eclipse projects. This document explains how to set up your new checkout in Eclipse.

**Starting up Eclipse**

Open Eclipse, if it is the first time you have used it click on the arrow icon to bring up the work environment:

To import the InterMine projects:

2. From the dialog select ‘General’ then ‘Existing Projects into Workspace’ and click Next.
3. Choose `git/intermine` (or whatever the root directory of your InterMine checkout is) as the root directory to import from.
4. A long list of projects will appear, leave them all selected and click Finish.
5. The workspace will take some time to build. If you are running eclipse on a new checkout of InterMine there initially be some errors. This is because some packages depend on code that is auto-generated from model XML files. To fix this you need to run some ant targets at the command line and refresh. You only need to do this the first time you set up Eclipse.
6. Run ant with no arguments in the following directories:
   - `intermine/objectstore/model/testmodel`
   - `intermine/integrate/model/fulldata`
   - `intermine/api/test`
   - `intermine/pathquery/main`
   - `intermine/api/model/userprofile`
   - `bio/test-all/dbmodel`
   Or just run as a single command:
   ```
   (cd intermine/objectstore/model/testmodel && ant clean-all && ant) && (cd intermine/integrate/model/fulldata && ant clean ... clean && ant build-db) && (cd intermine/objectstore/test && ant clean && ant)
   ```
Note: For the final directory you will first need to set up a properties file for the bio tests (see instructions running the bio tests) and may need to run ant clean build-db if prompted to do so.

7. Now select all projects in the Project Explorer and Refresh. There should no longer be any errors.

Debugging InterMine

1. start up tomcat with a port open for monitoring
2. Make a copy tomcat’s bin/startup.sh - call it debug_startUp.sh - make a version and defines the environment variable
   ```
   export JPDA_OPTS=-agentlib:jdwp=transport=dt_socket,address=<your host name>:8069,server=y,suspend=y
   ```
or, whatever port is free if 8069 is being used.
3. Replace the final line with
   ```
   exec "$PRGDIR"/"$EXECUTABLE" jpda start "$@
   ```
4. Go to eclipse and debug as a remote Java application. You just need to specify the host of Tomcat and the port that you listed. You’ll need to have a browser window or web service call to get things going.

Get the Software

The latest InterMine release is hosted on GitHub in the InterMine organization: https://github.com/intermine/intermine

1. Create an account at GitHub.
2. Visit the InterMine GitHub repository at https://github.com/intermine/intermine and click on the “Fork” button in the top right corner of the page.
3. Clone the forked InterMine repo to your local machine where git_username is the name of your GitHub account:
   ```
   $ git clone git@github.com:git_username/intermine.git
   ```
4. Check out the latest stable InterMine release from the ‘‘master’’ branch:
   ```
   $ cd intermine
   $ git checkout master
   ```
   (a) The other branches are used by InterMine developers for testing.
   (b) If you would like to use a specific release of InterMine, you can use one of our tags
5. To keep track of the original InterMine repo, assign the original repo to a remote called “upstream”
   ```
   $ git remote add upstream git@github.com:intermine/intermine.git
   ```
6. To pull changes in your local repository and merge them into your working files:
   ```
   $ git pull upstream
   ```
   You can also pull changes in your local repository without modifying your files
$ git fetch upstream

and merge any changes fetched into your working files

$ git merge upstream/master

All InterMine code is freely available under the open source LGPL license.

Get started

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 post-processing after data is loaded
• Deploy a webapp to query the data

Getting Started

We use git to manage and distribute source code. Download InterMine software from Get the Software and dependencies from System Requirements.

Creating a new Mine

Change into the directory you checked out the InterMine source code to and look at the sub-directories:

$ cd git/intermine
$ ls

The directories include:

intermine the core InterMine code, this is generic code to work with any data model.
bio code to deal specifically with biological data, it includes sources to load data from standard biological formats.
flymine the configuration used to create FlyMine, a useful reference when building your own Mine.
testmodel a non-biological test data model used for testing the core InterMine system.
imbuild InterMine’s ant-based build system, you shouldn’t need to edit anything here.

All configuration to create a new Mine is held in a directory in git/intermine, your Mine will depend on code in intermine, bio and imbuild. Any Mine needs to be a top level directory in your InterMine checkout.

There is a script for creating the Mine; in git/intermine run:

# in git/intermine
$ bio/scripts/make_mine MalariaMine

You will see the message: Created malariamine directory for MalariaMine

A malariamine directory has been created with several sub-directories, change into the directory and look what is created:
We will look at each of the sub-directories in much more detail later, they are:

**dbmodel** contains information about the data model to be used and ant targets relating to the data model and database creation.

**integrate** provides ant targets for loading data into the Mine.

**postprocess** ant targets to run post-processing operations on the data in the Mine once it is integrated.

**webapp** basic configuration and commands for building and deploying the web application

In addition there are two `default.intermine.xxx.properties` files which we won’t need to edit and a `project.xml` file. Most generated directories have a `project.properties` file and a short `build.xml` file, these are used by the InterMine build system.

**Project.xml**

The `project.xml` allows you to configure which data to load into your Mine. The automatically generated file has empty `<sources>` and `<post-processing>` sections:

```bash
$ less project.xml
```

There is a `project.xml` already prepared to define a new MalariaMine, copy it to this directory now and look at it:

```bash
$ cp ../bio/tutorial/project.xml .
$ less project.xml
```

**<sources>** The `<source>` elements list and configure the data sources to be loaded, each one has a `type` that corresponds to a directory in `git/intermine/bio/sources` or a subdirectory (the locations of sources to read are defined by `source.location` properties at the top of the file). These directories include 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 (more on this later).

`<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. 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 InterMine checkout includes a tar file with data to load into MalariaMine. These are real, complete data sets for "P. falciparum". We will load genome annotation from PlasmoDB, protein data from UniProt and GO annotation also from PlasmoDB.

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

```bash
$ cd
$ cp git/intermine/bio/tutorial/malaria-data.tar.gz .
$ tar -zxvf malaria-data.tar.gz
```

In your `malariamine` directory edit `project.xml` to point each source at the extracted data, just replace `DATA_DIR` with `/home/username` (or on a mac `/Users/username`). Do use absolute path.
$ cd ~/git/intermine/malariamine
$ sed -i 's/DATA_DIR/~/username/g' project.xml

For example, the uniprot-malaria source:

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

**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 maliaramine.properties file.

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

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

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

```bash
$ cd
$ cp git/intermine/bio/tutorial/malariamine.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 in Step 13.

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 prot other than 3306
# db.production.datasource.port=PORT_NUMBER
db.production.datasource.databaseName=malariamine
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 malariamine and items-malariamine postgres databases as specified in the malariamine.properties file:

```bash
$ createdb malariamine
$ createdb items-malariamine
```

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 MalariaMine, 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 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, we'll see how to do this later.
- All targets relating to the model for a Mine are in the dbmodel directory, go there now:

  ```
  $ cd ~/git/intermine/malariamine/dbmodel
  ```

  The core data model (and some extra model files) are defined in the project.properties file:

  ```
  core.model.path = bio/core
  ```

  You can view the contents of the core model:

  ```
  $ less ../../../bio/core/core.xml
  ```

  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.Integer"/>
  <reference name="sequence" referenced-type="Sequence" reverse-reference="proteins"/>
  <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"/>
  ...
  ```

  The model is generated from a core model XML file and any number of additions files. The first file merged into the core model is the so_additions.xml file. This XML file is generated from terms listed in the so_terms file. The build system creates classes corresponding to the Sequence Ontology terms:
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 an example ‘additions’ file for the UniProt source:

```
$ less ../../bio/sources/uniprot/uniprot_additions.xml
```

This defines extra fields for the Protein class which will be added to those from the core model. * Other model components can be included by specifying in the `dbmodel/project.properties` file, for example we include `bio/core/genomic_additions.xml` * The reverse-reference elements in definitions of some references and collections, this defines in the model that two references/collections are opposing ends of the same relationship. The value should be set to the name of the reference/collection in the referenced-type.

**Creating a database**

Now run the ant target to merge all the model components, generate Java classes and create the database schema, in `dbmodel` run:

```
# in malariamine/dbmodel
$ ant clean build-db
```

The clean is necessary when you have used the target before, it removes the build and dist directories and any previously generated model.

This target has done several things:

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

```
$ less build/model/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:

```
$ less build/model/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:

```
$ less build/gen/src/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 malariamine.properties as `db.production` - in our case malariamine. Log into this database and list the tables and the columns in the protein table:

```
Note: It may be necessary to switch to the user malariamine before continuing.
```

```
$ psql malariamine
malariamine=# \

```

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:

```shell
malariamine=# \d genesproteins
```

**Warning:** Running build-db will destroy any existing data loaded in the malariamine 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:

```shell
malariamine=# select key from intermine_metadata;
```

### Loading Data

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 ant in the integrate directory. You can specify one or more sources to load or choose to load all sources listed in the project.xml file. When you specify sources by name the order that they appear in project.xml doesn’t matter. Now load data from the uniprot-malaria source:

```shell
$ cd ../integrate
$ ant -Dsource=uniprot-malaria -v
```

The -v flag is to run ant in verbose mode, this will display complete stack traces 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
3. Calls the parser included in the uniprot source with the list of files, this reads the original XML and creates Items which are metadata representations of the objects that will be loaded into the malariamine database. These items are stored in an intermediate items database (more about Items later).
4. Reads from the items database, converts items to objects and loads them into the malariamine database.

This should complete after a couple of minutes, if you see an error message then see Troubleshooting tips.

If an error occurred during loading and you need to try again you need to re-initialise the database again by running clean build-db in dbmodel. This is only the case if dataloading actually started - if the following was displayed in the terminal:

```shell
[ant] load:
[ant]   [echo] Loading uniprot-malaria (uniprot) tgt items into production DB
[ant]   [echo] Loading uniprot-malaria (uniprot) tgt items into production DB
[ant]   [echo] Loading uniprot-malaria (uniprot) tgt items into production DB
```
A useful command to initialise the database and load a source from the integrate directory is:

```
$ (cd ../dbmodel; ant clean build-db) && ant -Dsource=uniprot-malaria
```

Now that the data has loaded, log into the database and view the contents of the protein table:

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

And see the first few rows of data:

```
malariamine# 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 Sequence-Features (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:

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

The same rows appear in the `sequencefeature` table:

```
malariamine# 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:

```
malariamine# 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:

```
malariamine# 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:

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

GFF3 files

GFF3 is a standard format used to represent genome features and their locations. It is flexible and expressive and defined by a clear standard - http://www.sequenceontology.org/gff3.shtml. An example of the file will load can be used to explain the format, each line represents one feature and has nine tab-delimited columns:

```
<table>
<thead>
<tr>
<th>seqid</th>
<th>source</th>
<th>type</th>
<th>start</th>
<th>end</th>
<th>score</th>
<th>strand</th>
<th>phase</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAL1</td>
<td>ApiDB</td>
<td>gene</td>
<td>183057</td>
<td>184457</td>
<td></td>
<td>-</td>
<td></td>
<td>ID=gene.46311;description=hypothetical</td>
</tr>
<tr>
<td>MAL1</td>
<td>ApiDB</td>
<td>mRNA</td>
<td>183057</td>
<td>184457</td>
<td></td>
<td>+</td>
<td></td>
<td>ID=mRNA.46312;Parent=gene.46311</td>
</tr>
<tr>
<td>MAL1</td>
<td>ApiDB</td>
<td>exon</td>
<td>183057</td>
<td>184457</td>
<td></td>
<td>-</td>
<td>0</td>
<td>ID=exon.46313;Parent=mRNA.46312</td>
</tr>
</tbody>
</table>
```

- **col 1**: “seqid” an identifier for a ‘landmark’ on which the current feature is located, in this case ‘MAL1’, a “P. falciparum” chromosome.
- **col 2**: “source” the database or algorithm that provided the feature
- **col 3**: “type” a valid SO 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_DIR/malaria/genome/gff/MAL1.gff3
```

The GFF3 source

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

Many elements can be configured by properties in project.xml, 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.

1.3. Get started
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)

In some cases specific code is required to deal with attributes in the gff file and any special cases. A specific source can be created to contain the code to do this and any additions to the data model necessary. For malaria gff we need a handler to switch which fields from the file are set as primaryId and secondaryId 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 primaryId.

The `malaria-gff` source is held in the `bio/sources/example-sources/malaria-gff` directory. Look at the `project.properties` file in this directory, 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
```

Look at the `MalariaGFF3RecordHandler` class in `bio/sources/example-sources/malaria-gff/main/src/org/intermine/bio/dataconversion`. This code changes which fields the `ID` and `Name` attributes from the GFF file have been assigned to.

$ less ~/git/intermine/bio/sources/example-sources/malaria-gff/main/src/org/intermine/bio/dataconversion/MalariaGFF3RecordHandler.java

---

**Loading GFF3 data**

Now load the `malaria-gff` source by running this command in `malariaamine/integrate`:

```
$ ant -Dsource=malaria-gff -v
```

This will take a few minutes to run. Note that this time we don’t run `build-db` in `dlmodel` 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 malariaamine
malariaamine#  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:
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 классAttribute` 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`

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 load the `malaria-chromosome-fasta` source by running this command in `malariamine/integrate`:

```
$ ant -Dsource=malaria-chromosome-fasta -v
```

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 MalariaMine

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”:

```
malariamine=# select id, primaryidentifier, secondaryidentifier, symbol, length , chromosomeid, chromosomelocationid, organismid from gene where primaryIdentifier = 'PFL1385c';
```

<table>
<thead>
<tr>
<th>id</th>
<th>primaryidentifier</th>
<th>secondaryidentifier</th>
<th>symbol</th>
<th>length</th>
<th>chromosomeid</th>
<th>chromosomelocationid</th>
<th>organismid</th>
</tr>
</thead>
<tbody>
<tr>
<td>83000626</td>
<td>PFL1385c</td>
<td>gene.33449</td>
<td>ABRA</td>
<td>2232</td>
<td>84017653</td>
<td>84018828</td>
<td>83000003</td>
</tr>
</tbody>
</table>

(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:

```
malariamine=# select id, primaryidentifier, secondaryidentifier, symbol, length , chromosomeid, chromosomelocationid, organismid from gene where primaryIdentifier = 'PFL1385c';
```

<table>
<thead>
<tr>
<th>id</th>
<th>primaryidentifier</th>
<th>secondaryidentifier</th>
<th>symbol</th>
<th>length</th>
<th>chromosomeid</th>
<th>chromosomelocationid</th>
<th>organismid</th>
</tr>
</thead>
<tbody>
<tr>
<td>83000626</td>
<td>PFL1385c</td>
<td>gene.33449</td>
<td>ABRA</td>
<td>2232</td>
<td>84017653</td>
<td>84018828</td>
<td>83000003</td>
</tr>
</tbody>
</table>

(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:

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

### Primary keys in MalariaMine

The keys used by each source are configured in the corresponding bio/sources/ directory.

For uniprot-malaria:

```
$ less ../../bio/sources/uniprot/resources/uniprot_keys.properties
```

And malaria-gff:

```
$ less ../../bio/sources/example-sources/malaria-gff/resources/malaria-gff_keys.properties
```
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.

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 _keys.properties files.

**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 primaryidentifier = 'PFL1385c';
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 and strain. 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 malariamine
malariamine# select * from organism;
```
From the `integrate` directory run the `entrez-organism` source:

```bash
$ ant -v -Dsource=entrez-organism
```

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:

```bash
$ psql malariamine
malariamine# select * from organism;
```

**Note:** 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:

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

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

```bash
$ ant -v -Dsource=update-publications
```

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:

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

Sometimes, especially with very large numbers of publications, this source will fail to fetch details correctly. Usually running it again will work correctly.

Occasionally erroneous PubMed ids are included from some sources and their details will not be updated, there is no good way to deal with this situation.

**Note:** 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.

**MalariaMine Post Processing**

The following `<post-process>` targets are included in the MalariaMine project.xml. The post-processes are run as a single stage of the build process. (see step 11.2 below for how to run the post-processing steps).

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.

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

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

### 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:

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

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 MalariaMine but you should always include the do-sources element.

### summarise-objectstore, create-search-index & create-autocomplete-index

These generate summary data and indexes used by the web application, see WebappConfig.

**Run the post-processing**

To run all the post-processing steps:
```
$ cd ../postprocess
$ ant -v
```

This will take a few minutes. When complete you can re-run the queries above to see what has been added.

Post-processing steps can also be run individually.
Building a Mine

So far we have created databases, integrated data and run post-processing with individual ant targets. InterMine includes a perl program called project_build that reads the project.xml definition and runs all of the steps in sequence. It also has the option of dumping the production database during the build and recovering from these dumps in case of problems.

Build complete MalariaMine

Build MalariaMine now using the project_build script, we will need a completed MalariaMine for the webapp.

Run the project_build script from your malariamine directory:

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

This will take ~15-30mins to complete.

Note: If you encounter an “OutOfMemoryError”, you should set your $ANT_OPTS variable, see Troubleshooting tips

Deploying the web application

Once you have read access to a production database, you can build and release a web application against it.

Configure

In the ~/.intermine directory, update the webapp properties in your malariamine.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 malariamine.properties file with correct information for the db.userprofile-production database:

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

2. Create the empty database:

$ createdb userprofile-malariamine

3. Build the database:
# in malariamine/webapp
$ ant build-db-userprofile

**Warning:** The build-db and build-db-userprofile commands rebuild the database and thus will delete any data.

This command creates the SuperUser account and loads the *default-template-queries.xml* file. You only need to build the userprofile database once.

### Deploying the webapp

**Tomcat**  
Tomcat is the webserver we use to launch InterMine webapps. Start Tomcat with this command:

```
# from the directory where tomcat is installed.
$ bin/startup.sh
```

Visit the Tomcat manager at [http://localhost:8080/](http://localhost:8080/). The username and password required to access the manager are `webapp.manager` and `webapp.password` as specified in *malariamine.properties*.

**Note:** There are extra steps to take if you are using Tomcat 7. See [Tomcat](#) for details.

**Webapp**  
Run the following command to release your webapp:

```
# in malariamine/webapp
$ ant default remove-webapp release-webapp
```

This will fetch the model from the database and generate the model java code, remove and release the webapp. The `default` target forces a rebuild of the `.war` file. If you’ve made updates, you may want to add `clean`, which removes temporary directories.

### Using the webapp

Navigate to [http://localhost:8080/malariamine](http://localhost:8080/malariamine) to view your webapp. The path to your webapp is the `webapp.path` value set in *malariamine.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

**Ant**

Anytime you run *ant* and something bad happens, add the verbose tag:

```
$ ant -v
```

This will give you more detailed output and hopefully a more helpful error message.
InterMine Documentation, Release

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.

Data files to integrate   All data required to build an InterMine is included in bio/tutorial/malaria-data.tar.gz. Copy this file to your local directory and extract from the archive.

cp bio/tutorial/malaria/malaria-data.tar.gz DATA_DIR
cd DATA_DIR
tar -zxvf malaria-data.tar.gz

Edit the project.xml file so that all occurrences of “DATA_DIR” point to your local data directory location.

Data sources

malaria-genome   The malaria genome as gff3 and fasta, originally downloaded from PlasmoDB

uniprot   UniProt XML with protein information and sequences from SwissProt and Trembl. Downloaded from: http://www.ebi.ac.uk/uniprot/database/download.html and filtered on taxon id 36329.


go_annotation   GO term assignments for “P. falciparum”. Downloaded from http://www.geneontology.org/

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. If Tomcat has run out of memory, it may not shut down correctly. Memory handling seems to have greatly improved with Tomcat 7 however.

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.

Figure 1.1: Header and footer of FlyMine website

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: `MINE_NAME/webapp/resources/webapp/model/images`. (If you don’t have a logo for your mine yet, you can still test updating your mine by using the FlyMine logo above.)

2. Deploy your webapp with this command:

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

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.

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

```shell
$ ant clean; ant default remove-webapp release-webapp
```

This removes all temporary directories so you are certain your new files are being used. The `clean-all` target removes temporary directories from all dependencies as well.

See *Ant* for a list of all ant targets.

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

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.

   ```shell
   $ emacs ~/.intermine/malariamine.properties
   ```

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

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

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

![Updated release version and subtitle](image)

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/malaria/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.

![Funding message in footer](https://github.com/intermine/intermine/blob/dev/intermine/webapp/main/resources/webapp/footer.jsp#L31)

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

```properties
# 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="Wellcome Trust">Wellcome Trust</a> and interoperates is funded by <a href="http://nihms.nih.gov/" target="_new" title="US National Institutes of Health">National Institutes of Health</a>
```

Here is the bit of code in footer.jsp that renders that message: [https://github.com/intermine/intermine/blob/dev/intermine/webapp/main/resources/webapp/footer.jsp#L31](https://github.com/intermine/intermine/blob/dev/intermine/webapp/main/resources/webapp/footer.jsp#L31)

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/malariamine.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 *Analyze*.

![Three boxes at the top of the home page](image)

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.

**InterMine, bio and mine /webapp** In Intermine there are 3 webapp projects: InterMine, bio and mine. You shouldn’t ever have to change the files in InterMine and bio, you’ll only ever update your mine’s files. When the webapp is compiled, the build system starts with the InterMine webapp project, then merges bio into that. Finally your mine’s webapp is added. The files and properties set in bio override any in the InterMine project. Your mine’s files and properties override any in bio or InterMine.

Therefore when you set `begin.listBox.example` in your properties file, it overrode the same property set in the InterMine properties file. This will be true of any property.

**Use your own**

The text and settings are configurable, but you may want a different layout for your home page.

1. Copy `begin.jsp` from `intermine/webapp` into your own webapp directory: `MINE_NAME/webapp/resources/webapp/model`.
2. Edit your `begin.jsp`
3. Redeploy your webapp to show your change
We saw in the previous section that properties override InterMine properties. The same holds true for JSP pages.

Note: Changes made to the home page, or whichever page you updated, will not be reflected in your custom copy. See *Home page* for more details on how to update your home page.

**Testmodel**

Create a file called *testmodel.properties*

- This file tells the testmodel application which databases to use.
- You will need to change the properties to your own database server and database username/password.
- Put it in `~/.intermine` with the following contents.

```properties
# Your database settings
db.userprofile-test.datasource.serverName=localhost
db.userprofile-test.datasource.databaseName=userprofile-testmodel
db.userprofile-test.datasource.user=USERNAME
db.userprofile-test.datasource.password=PASSWORD

db.unittest.datasource.serverName=localhost
db.unittest.datasource.databaseName=unittest
db.unittest.datasource.user=USERNAME
db.unittest.datasource.password=PASSWORD

project.sitePrefix=http://www.flymine.org/flymine
project.releaseVersion=test

webapp.deploy.url=http://localhost:8080
webapp.baseurl=http://localhost:8080
webapp.path=intermine-test
webapp.manager=TOMCAT_MANAGER
webapp.password=TOMCAT_PASSWORD
webapp.logdir=/var/log
superuser.account=SUPERUSER_EMAIL

project.rss=NEWS-FEED

# You don't need to change these
webapp.os.alias=os.unittest
webapp.userprofile.os.alias=osw.userprofile-test
project.standalone=true
```

Manually create the databases needed by the testmodel application. Execute:

```
$ createdb unittest
$ createdb userprofile-testmodel
```

Initialise the two databases:

```
$ cd testmodel/dbmodel
$ ant clean build-db
$ ant insert-data-unittest
$ cd ..;/webapp/main
$ ant build-db-userprofile
```

Build the testmodel web application and release to the running tomcat:
If you have previously released a webapp to /intermine-test you will need to remove it as well:

```
$ cd testmodel/webapp/main
$ ant default remove-webapp release-webapp
```

Browse to http://localhost:8080/intermine-test to view the web interface for the testmodel application.

## InterMine Tests

### Continuous Integration

We run all our tests on every commit using the Continous Integration service Travis-CI. 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 continuous 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

There are separate test projects for the main InterMine libraries: core objectstore code, the integration code and the web code. These are the packages/directories:

- intermine/objectstore/test
- intermine/integrate/test

Run the tests

Run the tests by changing to the appropriate directory and running `ant` with no arguments. For example:

# in intermine/objectstore/test/
$ ant

In this initial setup you may see some Java Exceptions, for diagnosis of common errors see: CommonErrors
View results

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

We aim to 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:

```bash
$ 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.

```properties
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/test-all/dbmodel:

```bash
$ ant clean build-db
```
Run the tests

Execute the tests, in `bio/test-all` run:

```bash
$ ant clean; ant
```

**Run a single test**

You can also run a test for an individual source by running the ant command with no arguments.

```bash
# in bio/sources/uniprot/test
$ ant
```

The test results will be located at `uniprot/test/build/test/results/index.html`. You can also run these as JUnit tests directly from Eclipse.

**Running the web application tests**

InterMine includes tests for running automated browser based user interface testing using Selenium. In particular the tests are meant to cover the main interface features of the generic web-application.

InterMine’s web applications tests are written in Python using unittest as the main test framework, selenium to interact with the Selenium webdriver and nose as a test runner.

The test suite can be found in the `intermine/testmodel/webapp/selenium/` directory.

**Environment Variables**

All tests run against a target which is the base URL of an InterMine instance.

```bash
# The base URL of the web application.
# Example: http://localhost:8080/intermine-demo
TESTMODEL_BASE
```

Credentials for a Gmail account are required to test OpenID authentication in `account-login-openid-test.py`:

```bash
# The username of a Gmail account
TESTMODEL_OPENID_NAME

# The password of a Gmail account
TESTMODEL_OPENID_PASSWORD
```

**Run the tests**

The tests are normally run as part of the CI test suite. They can also be run locally which is always a good idea when a new test is added or an existing test is modified.

To the run tests manually:

```bash
# in intermine/testmodel/webapp/selenium/
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirements.txt
$ nosetests
```
Developing test scripts

Selenium offers a Firefox plugin called Selenium IDE that can be used to record a user’s actions in the browser and then generate Selenium code in a variety of languages. While you may need to write code for more complex scenarios, the plugin can be a fast way to generate most of the work.

Python based test scripts should be placed in intermine/testmodel/webapp/selenium/test/ and their filename should end with “[filename]-test.py”. Test scripts in this directory are automatically included when nosetests is executed or when continual integration takes place.

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

Note: 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 http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html).

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 MalariaMine web application will be then available on
http://the_instance_public_DNS:8080/malariamine
To stop the web application:
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

HikariCP and InterMine settings

InterMine, starting with release 1.4, is using HikariCP as its default JDBC connection pool. If this is not available, InterMine will use the default postgresql distribution (PGPoolingDataSource).

HikariCP is a fast, reliable and lightweight connection pool. For details about the improvements it brings please refers to https://github.com/brettwooldridge/HikariCP.

From the InterMine user point of view, the main change is that the maximum number of connections allocated to a database, set in the properties files, is now reserved at the start and it is not any longer 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).

InterMine properties has been updated to deal with the change, but you could still need to edit the datasource.maxConnections properties specific for your mine, see “Which properties?” section below.

Note:

Reasonable settings for datasource.maxConnections can be seen in the files
• flymine/default.intermine.integrate.properties
• flymine/default.intermine.webapp.properties

with values of 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.

For more information about the properties files in InterMine, please refer to our Tutorial, and in particular to sections

1.3. Get started
Which properties?

The properties that you should check have a reasonable setting for a database maxConnections are, in order of priority:

1. YOUR_MINE.properties in your .intermine directory (if you set them there)
2. YOUR_MINE/default.intermine.integrate.properties (for building a mine)
3. YOUR_MINE/default.intermine.webapp.properties (for building a web application)

Some further information about the order of precedence for the setting of properties is available, for the web application, at http://intermine.readthedocs.org/en/latest/webapp/properties/web-properties/?highlight=overrides#overriding-properties

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:

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

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 the appropriate properties file

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

Further readings and references

https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing
https://groups.google.com/forum/#!forum/hikari-cp

Data Model

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 through 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_term_list.txt).
- Only the terms listed in so_term_list.txt 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_list.txt is augmented by the contents of intermine/bio/core/core.xml and intermine/bio/core/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.

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.

There are effectively two types of reference and two types of collection, depending on the type or presence of a reverse 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.

Many to one relationship this is where a reference has a reverse-relationship that is a collection, or where a reference does not have a reverse-relationship.

One to many relationship this is where a collection has a reverse-relationship that is a reference. This kind of relationship is a side-effect of a many to one relationship, and cannot be written to from this end. All alterations should be made on the many to one relationship instead.

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 from both sides.

A short example

```xml
<?xml version="1.0"?>
<model name="testing" package="org.intermine.model.bio">
  <class name="Protein" is-interface="true">
    <attribute name="name" type="java.lang.String"/>
    <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.

**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

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChromosomeLocation</td>
<td>Chromosome Location</td>
</tr>
<tr>
<td>shortName</td>
<td>Short Name</td>
</tr>
<tr>
<td>Organism</td>
<td>Organism</td>
</tr>
<tr>
<td>name</td>
<td>Name</td>
</tr>
<tr>
<td>Organism.shortName</td>
<td>Organism &gt; Short Name</td>
</tr>
</tbody>
</table>

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:

```xml
<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

For this, 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/resources/webapp/WEB-INF/soClassName.properties` used to generate readable names using the SO term a class represents
- `bio/webapp/resources/webapp/WEB-INF/bioClassNames.properties` used to map non-SO classes to readable names
- `bio/webapp/resources/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:

```
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, an 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. eg:
  ```
  <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 represented by a string. eg:
  ```
  <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 path is represented by a string. eg:
  ```
  <c:out value="${imf:formatFieldStr(pathString, INTERMINE_API, WEBCONFIG)}"/>
  ```

The values `INTERMINE_API` and `WEBCONFIG` are automatically available within jsp's 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, e.g. 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:

```javascript
var className = ??;
var displayName = $MODEL_TRANSLATION_TABLE[className].displayName;
```

And for fields of this class:

```javascript
var fieldName = ??;
var fieldDisplayName = $MODEL_TRANSLATION_TABLE[className].fields[fieldName]
```

**Querying over genomic ranges**

**Note:** Starting with InterMine 1.4 Bioseg is no longer required if you are using Postgres version 9.2 or later.

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 index (either bioseg or the native Postgres one) 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.

**Install BIOSEG (Postgres versions 9.1 and older only)**

`BIOSEG` is a custom index type that you need to add to postgres. See [Range Queries](#) for details

**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:

```xml
<post-process name="create-location-range-index"/>
```

If you are using an older version of Postgres, this task will create a BIOSEG index. Otherwise it will use a native Postgres 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.

```xml
<post-process name="create-overlap-view" />
```

Now any queries on the `overlappingFeatures` collections will use this view and the new index.
Database

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 at bio/scripts/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 libblog-handler-perl libblog-report-perl
```

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:

```
bio/scripts/DataDownloader/config
```

The yaml file of your mine is where data download script reads the instructions.

Running

To run a set of data downloads, the following call should suffice:

```
perl bio/scripts/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 bio/scripts/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`:

```perl
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',
    }],
};
```

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:
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.

Data Sources

Contents

Data Source Library

This page lists the current sources available for use in InterMine. All the sources here are found in bio/sources. Look at flymine/project.xml for examples of how to use these sources.

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 identifier, these are the default values and you do not need to update the configuration file.

<table>
<thead>
<tr>
<th>parameter</th>
<th>definition</th>
<th>possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>typeAnnotated</td>
<td>class of what is being annotated</td>
<td>gene (default) or protein</td>
</tr>
<tr>
<td>identifier</td>
<td>which field to set</td>
<td>primaryIdentifier (default), symbol, or primaryAccession</td>
</tr>
<tr>
<td>readcolumn</td>
<td>which column to use for identifier</td>
<td>primaryAccession (default) or symbol</td>
</tr>
</tbody>
</table>

# an example entry
7165.typeAnnotated=protein
7165.identifier=primaryAccession

How to load the data into your mine  project XML example

```xml
<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

```xml
<source name="go" type="go">
    <property name="src.data.file" location="/data/go-annotation/gene_ontology.obo"/>
</source>
```

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.

Homologue Data Sources  InterMine comes with several data converter for homologue data, e.g. TreeFam, PANTHER, OrthoDB, Homologene, etc. Follow the instructions below to include these datasets in your InterMine.

Treefam

3See http://www.geneontology.org/GO.format.gaf-2_0.shtml for column description

1.5. Database
Data  ftp://ftp.sanger.ac.uk/pub/treefam/release-7.0/MySQL

Download two tables:

- genes.txt.table
- ortholog.txt.table

Project XML

```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
2. project.xml

```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


Unzip the files and put them in the same directory.

Project XML

```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

```xml
<source name="panther" type="panther">
  <property name="src.data.dir" location="/DATA/panther"/>
  <property name="panther.organisms" value="7227"/>
</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.

<table>
<thead>
<tr>
<th>Gene ID</th>
<th>Homologue ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSG00000253023</td>
<td>ENSMUSG00000088328</td>
</tr>
<tr>
<td>ENSG00000238364</td>
<td>ENSMUSG00000088728</td>
</tr>
</tbody>
</table>

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
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Query>
<Query virtualSchemaName = "default" formatter = "TSV" header = "0" uniqueRows = "0" count = "" datasetConfigVersion = "0.6">
    <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

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.

4. Create a sub directory `/DATA_DIR/idresolver/` as file root path and add symbolic links to the two files.

   ```bash
   $ 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`

   ```bash
   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:
We use WormMart but are happy to hear of a better source for worm identifiers.

Here are the project XML entries used by FlyMine:

https://github.com/intermine/intermine/blob/dev/flymine/project.xml#L36-L48

### 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>
<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

1.5. Database
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

```xml
<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

```xml
<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

```xml
<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

```xml
<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:

```plaintext
# 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

```xml
<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:

```xml
<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:

TAXON_ID_uniprot_sprot.xml
TAXON_ID_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

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>taxonomy</td>
<td>e.g. 9606 for human</td>
</tr>
<tr>
<td>reviewed</td>
<td>yes for swiss prot, no for trembl</td>
</tr>
<tr>
<td>compress</td>
<td>if yes, zipped</td>
</tr>
</tbody>
</table>

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**

```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>
```

<table>
<thead>
<tr>
<th>property</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>creategenes</td>
<td>if TRUE, process genes</td>
<td>true</td>
</tr>
<tr>
<td>creatego</td>
<td>if TRUE, process GO annotation</td>
<td>false</td>
</tr>
<tr>
<td>allowduplicates</td>
<td>if TRUE, allow proteins with duplicate sequences to be processed</td>
<td>false</td>
</tr>
<tr>
<td>loadfragments</td>
<td>if TRUE, load all proteins even if isFragment = true</td>
<td>false</td>
</tr>
<tr>
<td>loadtrembl</td>
<td>if FALSE, not load trembl data for given organisms, load sprot data only</td>
<td>true</td>
</tr>
</tbody>
</table>

**FASTA**  This source loads FASTA data for isoforms. The UniProt entry is does not contain the sequences for isoforms.


```xml
<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_varsplice.fasta"/>
  <property name="fasta.sequenceType" value="protein"/>
  <property name="fasta.loaderClassName" value="org.intermine.bio.dataconversion.UniProtFastaLoaderTask"/>
</source>
```

**UniProt keywords**  Loads the names for the UniProt keywords contained in the main UniProt converter.

Protein Data Bank

Types of data loaded  proteins, protein structures

How to load the data into your mine  project XML example

```xml
<source name="pdb" type="pdb">
  <property name="src.data.dir" location="/data/pdb"/>
  <property name="pdb.organisms" value="7227"/>
</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

```xml
<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

```xml
<!-- 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"/>
</source>
```
You’ll also need the DTD file here:
ftp://ftp.ebi.ac.uk/pub/databases/interpro/match_complete.dtd
Put it in your $MINE/integrate directory.

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


Unzip both files, save gene2pubmed under a directory named pubmed, e.g. DATA_DIR/pubmed. It’s suggested to save gene_info in a different directory, e.g. DATA_DIR/ncbi-gene, but you can always save both in pubmed directory, see how to config below.

How to load the data into your mine After code refactory and optimization, the current PubMed convertor will make use of id resolver to parse gene information (see how to setup Id Resolvers), whereafter the infoFile property was removed from the config.

If gene2pubmed is the only file in DATA_DIR/pubmed directory, you can remove src.data.dir.includes property, but do keep it if you place gene2pubmed and gene_info in pubmed dir at the same time.

project XML example

```xml
<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>
```

project XML example for InterMine 1.1 and older

```xml
<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"/>
  <property name="infoFile" location="DATA_DIR/ncbi-gene/gene_info"/>
</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, fetch details from the Entrez web service and fill in Publication objects.

Types of data loaded None, the publication 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

```xml
<source name="update-publications" type="update-publications" dump="true">
  <property name="src.data.file" location="build/publications.xml"/>
  <!-- <property name="cacheDirName" value="build/"/> -->
  <!-- <property name="loadFullRecord" value="true"/> -->
</source>
```

properties:
1. cacheDirName - the directory to generate publication cache files, default value “build/”
2. loadFullRecord - load MeSH terms and abstract, value “true”/”false”

Variation

Ensembl SNP  Load SNP data from a downloaded data file.

Types of data loaded  SNPs

How to download the data  First you will need the variation data from Ensembl, which are available via MySQL databases. Download the Ensembl file from their FTP site:

ftp://ftp.ensembl.org

Example:


Download and unzip the file.

How to load the data into your mine

Add Ensembl snp to the list of datasources to be integrated

```xml
<source name="ensembl-snp" type="ensembl-snp">
  <property name="src.data.dir" location="/data/variation/ensembl/current"/>
  <property name="ensembl-snp.includes" value="Homo_sapiens_incl_consequences.gvf"/>
</source>
```

Run InterMine build  Run a build. The entry in `project.xml` will instruct the build process to load the GVF file you downloaded in the previous step into the database. For example, run this command in `MINE_NAME/integrate`:

```
$ ant -v -Dsource=ensembl-snp
```

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:


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>
```

Run InterMine build  Run a build. The entry in project.xml will instruct the build process to load the XML files you created in the previous step into the database. For example, run this command in MINE_NAME/integrate:

```
$ ant -v -Dsource=my-data-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

1.5. Database
**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:

```xml
<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:

```xml
<source name="chado-db-flybase-dmel" type="chado-db">
  ...<property name="processors" value="org.intermine.bio.dataconversion.FlyBaseProcessor"/>
  ...
</source>
```

**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:

```xml
<source name="chado-db-some-database" type="chado-db">
  ...
  <property name="processors" value="org.intermine.bio.dataconversion.ChadoSequenceProcessor org.intermine.bio.dataconversion.StockProcessor"/>
  ...
</source>
```
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:

<table>
<thead>
<tr>
<th>gene1_feature_id</th>
<th>relation_type</th>
<th>protein1_feature_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene2_feature_id</td>
<td>relation_type</td>
<td>protein2_feature_id</td>
</tr>
<tr>
<td>gene2_feature_id</td>
<td>relation_type</td>
<td>protein1_feature_id</td>
</tr>
<tr>
<td>gene2_feature_id</td>
<td>relation_type</td>
<td>protein2_feature_id</td>
</tr>
</tbody>
</table>

(Assuming the unlikely case where two genes are related to two proteins)

If subjectFirst is false we get results like:

<table>
<thead>
<tr>
<th>gene1_feature_id</th>
<th>relation_type</th>
<th>protein1_feature_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene2_feature_id</td>
<td>relation_type</td>
<td>protein2_feature_id</td>
</tr>
<tr>
<td>gene2_feature_id</td>
<td>relation_type</td>
<td>protein1_feature_id</td>
</tr>
<tr>
<td>gene2_feature_id</td>
<td>relation_type</td>
<td>protein2_feature_id</td>
</tr>
</tbody>
</table>
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:

```properties
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

```xml
<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

```bash
$ cd MINE_NAME/integrate
$ (cd ../dbmodel && ant build-db -v); ant -Dsource=chado-db -v
```

See [Database Building](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](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](InterMine_1.8), you can instead directly define the schema in the properties of the database in your properties file, like
Ensembl Core  This page describes how to load Ensembl core data into your InterMine-bio database.

Generate Items XML file

Ensembl Data  First you will need the data from Ensembl, which are available via MySQL databases. Download the Ensembl MySQL database and create the database locally:


for example: download homo_sapiens_core_70_37 to a local directory, unzip all gz files, and load it to your MySQL database

    # create a new db in MySQL
    $ mysql -u DB_USER -p
    mysql> create database homo_sapiens_core_70;

    # load data into db
    $ mysql -u DB_USER -p homo_sapiens_core_70 < homo_sapiens_core_70_37.sql
    $ mysqlimport -u DB_USER -p homo_sapiens_core_70 -L *.txt -v

Update <MINE_NAME>.properties  Add the location of the downloaded Ensembl MySQL databases to your mine properties file, for example:

    # core database
db.ensembl.9606.core.datasource.serverName=SERVER_NAME
    # port: uncomment the next line if use different port other than 3306
    # db.ensembl.9606.core.datasource.port=PORT_NUMBER
    db.ensembl.9606.core.datasource.databaseName=homo_sapiens_core_70
    db.ensembl.9606.core.datasource.species=homo_sapiens
    db.ensembl.9606.core.datasource.user=DB_USER
    db.ensembl.9606.core.datasource.password=DB_PASSWORD

These properties are used by the Perl script.

Install Perl modules

Ensembl  InterMine’s Ensembl converter uses Ensembl’s Perl API. Follow Ensembl’s instructions for how to install the necessary Perl modules:

    http://www.ensembl.org/info/docs/api/api_installation.html

InterMine  You will also need to install InterMine’s Perl modules, see Perl

Run Script to Generate Items XML  Run this command in /bio/scripts

1.5. Database
$ ./ensembl.pl [Release Version] MINE_NAME TAXONID DATA_DESTINATION

for example:
$ ./ensembl.pl flymine 7165 /data/ensembl/current

Load XML file into InterMine database

Add Ensembl core to the list of datasources to be integrated  This is located in the project.xml file, and it should look something like:

```xml
<source name="ensembl" type="ensembl-core">
  <property name="src.data.dir" location="/MY_DATA_DIR/ensembl"/>
</source>
```

When you run a database build, every XML file in the directory specified will be processed and loaded into the database.

Run InterMine build  Run a build. The entry in project.xml will instruct the build process to load the XML files you created in the previous step into the database. For example, run this command in MINE_NAME/integrate:

$ ant -v -Dsource=ensembl

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

```xml
<source name="flybase-dmel-gene-fasta" type="fasta">
  <property name="fasta.taxonId" value="7227"/>
  <property name="fasta.dataSetTitle" value="FlyBase fasta data set for Drosophila melanogaster"/>
  <property name="fasta.dataSourceName" value="FlyBase"/>
  <property name="fasta.className" value="org.intermine.model.bio.Gene"/>
  <property name="fasta.classAttribute" value="primaryIdentifier"/>
  <property name="fasta.includes" value="dmel-all-gene-*.fasta"/>
  <property name="src.data.dir" location="/DATA/flybase/fasta/current"/>
</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
classAttribute | name of dataset | determines name of datasource object
classAttribute | name of data file | these data will be loaded into the database
classAttribute | name of Java file that will process the fasta files | this data file will be loaded into the database
type | class name | type of sequence to be loaded
sequenceType | class name | only use if you have created a custom fasta loader

**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

```bash
# example GFF3 file
MAL1 ApiDB gene 183057 184457 . - . ID=gene.46311;description=hypothetical protein;Name=PFA0210c
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**

**Note:** 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. For more advanced processing, you will have to write your own GFF3 parser. See below for details.
# gff_config.properties example for E. coli gff3 attributes

511145.terms=gene,exon # feature types to load, e.g. load gene and exon for E. coli
511145.excludes=CDS # comma-separated list of feature types to exclude
511145.attributes.Dbxref.EcoGene=primaryIdentifier # use Dbxref EcoGene field as primaryIdentifier
511145.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

You can edit the main gff_config.properties file in the bio/core package, or edit the config file in your new data source – in main/resources/<data-source>_config.properties.

Parent relationship The parent-child relationship between features can also be handled automatically if you set it up properly. Take MalariaGFF3RecordHandler for example:

```java
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:

```xml
<property name="example-gff3" type="gff">
    <property name="gff3.taxonId" value="7227"/>
    <property name="gff3.seqClsName" value="Chromosome"/>
    <property name="src.data.dir" location="/DATA/*.gff3"/>
</source>
```

Here are the descriptions of the properties available:

<table>
<thead>
<tr>
<th>property</th>
<th>example</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>gff3.seqClsName</td>
<td>Chromosome</td>
<td>the ids in the first column represent Chromosome objects, e.g. MAL1</td>
</tr>
<tr>
<td>gff3.taxonId</td>
<td>36329</td>
<td>taxon id of malaria</td>
</tr>
<tr>
<td>gff3.dataSourceName</td>
<td>PlasmoDB</td>
<td>the data source for features and their identifiers, this is used for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DataSet (evidence) and synonyms.</td>
</tr>
<tr>
<td>gff3.seqDataSource</td>
<td>PlasmoDB</td>
<td>the source of the seqids (chromosomes) is sometimes different to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>features described</td>
</tr>
<tr>
<td>gff3.dataSetTitle</td>
<td>PlasmoDB P. falciparum genome</td>
<td>a DataSet object is created as evidence for the features, it is linked to a DataSource (PlasmoDB)</td>
</tr>
</tbody>
</table>

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:

```java
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:

```java
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**

  - where FB20XX_XX = the current FlyBase release

- **zfin-identifiers** [http://zfin.org/downloads/ensembl_1_to_1.txt](http://zfin.org/downloads/ensembl_1_to_1.txt)

- **sgd-identifiers** [http://downloads.yeastgenome.org/curation/chromosomal_feature/SGD_features.tab](http://downloads.yeastgenome.org/curation/chromosomal_feature/SGD_features.tab)

- **wormbase-identifiers** query wormbase biomart webservice


**How to load the data into your mine**  project XML example
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 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

```xml
<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

```xml
<source name="so" type="so">
  <property name="src.data.file" location="./bio/sources/so/so.obo"/>
</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

```xml
<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

```xml
<source name="update-data-sources" type="update-data-sources">
    <property name="src.data.file" location="build/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 $MINE/integrate/datasets.xml. Use FlyMine’s datasets.xml as an example: https://github.com/intermine/intermine/blob/dev/flymine/integrate/datasets.xml

How to load the data into your mine  project XML example

```xml
<source name="flymine-static" type="intermine-items-xml-file">
    <property name="src.data.file" location="datasets.xml"/>
</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 bio/sources/fly.

- affy-probes
- anoest
- anopheles-identifiers
- anoph-expr
- arbeitman-items-xml
- bdgp-clone
- bdgp-insitu
- drosdel-gff
- drosophila-homology
- fly-anatomy-ontology
- flyatlas
- fly-development-ontology
- fly-fish
- fly-misc-cvterms
- flyreg
• flyrnai-screens
• homophila
• long_oligo
• protein_structure
• redfly
• rai
• tiffin
• tiffin-expression
• tiling_path

See FlyMine for more information about these datasets. Look at flymine/project.xml for examples of how to use these sources.

Other loaders

These loaders were written by InterMine users.
• CHEBI
• Disease ontology
• haem-atlas
• HGNC
• HuGE
• Mammalian phenotypes
• pride
• stitch
• huge gwas

Writing your own data source

The aim of this tutorial is to create a new source. Often this will be to import data from an InterMine Items XML format file that you create, though other types of source can also be created (e.g. a source that extends InterMine’s existing GFF3 importer.

There are three parts to creating a new source:

1. Create the directory structure in InterMine that contains files that describe the source.
2. Configure the mine to use this source (make an entry in your project.xml).
3. Write code to parse the source. You can either do this in InterMine directly, by extending the DataConverter class, or you can use some other language to generate a standalone InterMine Items XML file, and set have.file.xml.tgt = true in the source’s project.properties file. See this page for more information on the InterMine Items XML file format and links to language-specific APIs (Perl, Python, etc.) that can help create it.
Create source files

Run make_source script This script creates the basic skeleton for a source. It should be run in the top level directory of an InterMine checkout, like this:

$ ./bio/scripts/make_source <source-name> <source-type>

Possible source types 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 FileConverter will be needed. The make_source script will create a skeleton FileConverter in bio/sources/<source-name>/main/src/org/intermine/bio/dataconversion. Edit this code to process the particular file you need to load, using the internal Java Items API to create and store items to the database.

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:

<property name="src.data.file" location="/some/directory/objects_in_intermine_format.xml"/>

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 ANT_OPTS environment variable).

db This source reads directly from a relational database, it will generate a skeleton DBConverter in bio/sources/<source-name>/main/src/org/intermine/bio/dataconversion. To connect to the database you need to add properties in xxxmine.properties with the prefix db.sourcename. This is tested for PostgreSQL and MySQL.

Common properties:

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
Oracle Data Source  You can load data from Oracle directly into an InterMine instance. The JDBC driver is already in the classpath: `intermine/objectstore/main/lib ojdbc14.jar`

Database Properties  Add this to your mine properties file, these are the values that are going to be passed to Oracle in the connection string:

```
db.mysourcedb.datasource.class=oracle.jdbc.pool.OracleDataSource
db.mysourcedb.datasource.dataSourceName=db.sgd
db.mysourcedb.datasource.serverName=HOSTNAME
db.mysourcedb.datasource.databaseName=XE
db.mysourcedb.datasource.user=USER
db.mysourcedb.datasource.password=PWD
db.mysourcedb.datasource.maxConnections=10
db.mysourcedb.driver=oracle.jdbc.driver.OracleDriver
db.mysourcedb.datasource.driverType=thin
db.mysourcedb.datasource.portNumber=1521
db.mysourcedb.platform=Oracle
```

Data Source  Create a new source with a `db source_type`, eg:

```
$ ./bio/scripts/make_source MySource db
```

Add this source to your project.xml file. The value, `mysourcedb` in this example, tells the build system to get the database information from the `db.mysourcedb` entry in your properties file.

```
<source name="mysource" type="mysource">
    <property name="source.db.name" value="mysourcedb"/>
</source>
```

Data Converter  You can now connect to Oracle in your source. In your converter, get the connection to the database

```
// a database has been initialised from properties starting with db.sgd
Connection connection = getDatabase().getConnection();
```

Run a query to retrieve your data

```
String query = "SELECT feature_no, dbxref_id FROM feature WHERE feature_type = 'gene';"
Statement stmt = connection.createStatement();
ResultSet res = stmt.executeQuery(query);
```

Iterate through results and convert each row to an InterMine object.

```
while (res.next()) {
    // create a gene
    Item item = createItem("Gene");

    // set gene fields from results
    item.setAttribute("primaryIdentifier", res.getString("dbxref_id"));
    item.setAttribute("secondaryIdentifier", res.getString("feature_name"));

    // store item in database
    store(item);
}
```

It is good practice to put the properties that won’t change in `MINE_NAME/default.intermine.integrate.properties` and those that may change (`serverName`, `databaseName`, `user`, `password`) in `~/.intermine/MINE_NAME.properties`.

1.5. Database
The `db` value has to match the `"source.db.name"` in your project XML entry, for example:

```xml
# project XML
<source name="chado-db-flybase-dmel" type="chado-db">
  <property name="source.db.name" value="flybase"/>
  ...
</source>
```

```properties
# flymine.properties

db.flybase.datasource.class=com.zaxxer.hikari.HikariDataSource
db.flybase.datasource.dataSourceClassName=org.postgresql.ds.PGSimpleDataSource
db.flybase.datasource.dataSource=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
```

gff  Create a gff source to load genome annotation in GFF3 format. This creates an empty `GFF3RecordHandler` in `bio/sources/<source-name>/main/src/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 for more information on integrating GFF3.

obo  Create a obo source to load ontology in BO format.

**Additions file**  Update the file in the source folder 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>
```

**Note:** If all the data you wish to load is already modelled in InterMine then you don’t need an additions file.

**Properties** Any properties you define in a source entry in your mine’s project.xml will be set on 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.DBCConverter`
- `org.intermine.dataconversion.DirectoryConverter`
- `org.intermine.dataconversion.FileConverter`
- `org.intermine.postprocess.PostProcessor`

For instance, if you have the source entry

```xml
<source name="my-new-source-name" type="my-new-source">
  <property name="fooFile" location="/some/directory/objects_in_intermine_format.xml"/>
  <property name="bar.info" location="baz"/>
</source>
```
in your project.xml file and a class that extends org.intermine.postprocess.PostProcessor, then before post-processing the following methods will be called on that class with these parameters

```java
myPostProcessor.setFooFile(new File("/some/directory/objects_in_intermine_format.xml");
myPostProcessor.setBarInfo("baz");
myPostProcessor.setBazMoreInfo("hello-world");
```

**Keys file**  Within the 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 proteins by their (UniProt) primaryAccession attribute so we define that this source should use the key:

```
Protein=key_primaryacc
```

Note that we don’t expect any other data sources to provide interesting features so we don’t need to integrate them - no key is defined. The possible keys are defined in `dbmodel/resources/genomic_keyDefs.properties`, new keys can be added if necessary.

### Including your source in a Mine

**Project XML**  In the project.xml file, in the root of your mine directory (e.g. /malariamine), the following entries should be added and altered accordingly:

```xml
<source name="new-source-name" type="new-source">
  <property name="src.data.file" location="my_data_dir/example.xml"/>
</source>
```

If you have more than one file you can set this up to point at a `'directory'`:

```xml
<source name="new-source-name" type="new-source">
  <property name="src.data.dir" location="my_data_dir/source_files"/>
</source>
```

The first line defines the name you wish to give to the of the source and the type - the name of the directory in `bio/sources`. The second line defines the location and name of the data file.

If you are using data from a database:

```xml
<source name="new-source-name" type="new-source">
  <property name="source.db.name" value="db.NAME"/>
  ...
</source>
```

The value of `source.db.name` must match the value set in the MINE_NAME.properties file.

**Run build-db**  Create the database as usual. The source should now be included when building the mine.

**Note:**  Unless the ‘clean’ is run (which deletes the build directory) in `MINE_NAME/dbmodel` any changes will append to the current model structure and any unwanted classes/attributes will remain.
InterMine Items XML

InterMine items XML is a generic format that encodes data that matches InterMine class definitions.

Unfortunately, there is no document giving the formal specification for this format yet (please ping us on the mailing list or elsewhere if you need this). However, an example and some information is below

```xml
<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:

- bio/scripts/intermine_items_example.pl
- RedFly testing objects file at bio/sources/flymine/redfly/test/resources/RedFlyGFF3RecordHandlerTest.xml

Datatypes

The data formats required for attributes in InterMine Items XML need proper documentation. However, 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:

```java
Item gene = createItem("Gene");
```

Store an item (or collection of items) to the target items database:

```java
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.

```java
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.

```java
String orgIdentifier = organism.getIdentifier();
gene.setReference("organism", orgIdentifier);
```

Or:

```java
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:

```perl
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:
Example In `bio/scripts` there is a longer example: [source:trunk/bio/scripts/intermine_items_example.pl 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 is from the `bio/scripts/` directory

Example command line: .. code-block:: perl

   ./intermine_items_example.pl "FlyMine" 5833 ../../flymine/dbmodel/build/model/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


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).

<table>
<thead>
<tr>
<th>unique gene identifier</th>
<th>symbol, name, ensembl ID ...</th>
<th>pax6, paired box gene 6 ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGI:97490</td>
<td>MGI:97490</td>
<td>MGI:97490</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>1212</th>
<th>10090</th>
<th>18508</th>
<th>Pax6</th>
<th>7305369 NP_038655.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1212</td>
<td>10116</td>
<td>25509</td>
<td>Pax6</td>
<td>6981334 NP_037133.1</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Resolver</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlyBaseIdResolverFactory</td>
<td>flybase chado db, for “D.melanogaster” only</td>
<td>ftp://ftp.flybase.net/releases/current/psq1</td>
</tr>
<tr>
<td>WormBaseChadoIdResolverFactory</td>
<td>wormbase chado db, for “C.elegans” only</td>
<td>modENCODE specific</td>
</tr>
<tr>
<td>ZfinIdentifiersResolverFactory</td>
<td>zebrafish ids</td>
<td><a href="http://zfin.org/downloads/identifiersForIntermine.txt">http://zfin.org/downloads/identifiersForIntermine.txt</a></td>
</tr>
<tr>
<td>RgdIdentifiersResolverFactory</td>
<td>rat ids</td>
<td>ftp://rgd.mcw.edu/pub/data_release/GENES_RAT.txt</td>
</tr>
<tr>
<td>HgncIdResolverFactory</td>
<td>HGNC human gene ids</td>
<td><a href="http://www.genenames.org/cgi-bin/hgnc_downloads.cgi">http://www.genenames.org/cgi-bin/hgnc_downloads.cgi</a></td>
</tr>
<tr>
<td>EnsemblIdResolverFactory</td>
<td>Ensembl id</td>
<td>customised</td>
</tr>
<tr>
<td>HumanIdResolverFactory</td>
<td>human ids</td>
<td>customised</td>
</tr>
</tbody>
</table>

**Using ID Resolvers in InterMine data converters**

Many data converters use the Entrez (NCBI) Gene ID resolver:

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:
<table>
<thead>
<tr>
<th>Resolver</th>
<th>Symbolic link</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntrezGeneIdResolverFactory</td>
<td>entrez</td>
</tr>
<tr>
<td>WormBaseChadoIdResolverFactory</td>
<td>wormid</td>
</tr>
<tr>
<td>ZfinIdentifiersResolverFactory</td>
<td>zfin</td>
</tr>
<tr>
<td>MgiIdentifiersResolverFactory</td>
<td>mgi</td>
</tr>
<tr>
<td>RgdIdentifiersResolverFactory</td>
<td>rgd</td>
</tr>
<tr>
<td>HgncIdResolverFactory</td>
<td>hgnc</td>
</tr>
<tr>
<td>EnsemblIdResolverFactory</td>
<td>ensembl</td>
</tr>
<tr>
<td>HumanIdResolverFactory</td>
<td>humangene</td>
</tr>
</tbody>
</table>

In the data converter, the ID resolver is given an identifier. The resolver then looks in the map for the identifier.

<table>
<thead>
<tr>
<th>number of matches</th>
<th>returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>new identifier</td>
</tr>
<tr>
<td>&gt;1</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**Using ID Resolvers in your data converters**

A factory will find data root path from `~/.intermine/MINE_NAME.properties`, path needs to be absolute.

```java
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 `EntrezGenIdResolverFactory`

```java
private final String propKey = "resolver.file.rootpath";
private final String resolverFileSymbo = "entrez";
```

As for database case, e.g. flybase chado

```java
# 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`

```java
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

```java
7227.primaryIdentifier.xref=FLYBASE
```

If you don’t want to strip the prefix from the identifier, use this config:
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 written 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
resolvertasmanAndClassName(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 createFromDb method.

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 multiple files, but one can always add them and parse them one by one.
// 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.
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 translates 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**

We recommend running long builds from the `bio/scripts/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 ant to avoid memory problems encountered when running many Java task sequentially from ant. 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*

Run the build script from the mine directory:

```
$ ../bio/scripts/project_build -l -t -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:

```xml
<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)
- `-r` attempt to restart just after the last dump point _without_ loading a dump (see note below)
- `-b` run build-db before starting build and drop any existing backup databases (created when using the `-t` flag)
-n  parse files and report actions, but don’t execute anything
-V  set the release number to pass to ant (as -Drelease=release_number) 4.
-D  set the destination database for the completed build; the database will be copied to this name in the same postgres server that the build used
-a  set the list of actions (ie. source integrations or post-processes) to perform - the list must be a subset of the sources/postprocesses in the project.xml file

• The -l and -r operate as usual.
• To run all steps starting at <some_action> use a dash after the action name: ‘‘-a <some_action>-‘‘
• To perform only the final dump use: ‘‘-a final-dump‘‘
• To refer to dump step and skip the corresponding action, use action name with “-dump” appended. eg.
  – ‘‘-a fly-fish-dump-‘‘ - dump the fly-fish source and continues integrating
  – ‘‘-a fly-fish-dump,flymine-static,create-utr-references,final-dump‘‘ - do just those steps

Dump files take the name dump_file_prefix.final. These dumps can be used by project_build to restart a build process after a previous problem. Running project_build with ‘‘-l‘‘ will reload the latest dump (if any) with dump_file_prefix exist 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:

$ cd MINE_NAME/dbmodel/
$ ant build-db

Warning: Running the build-db target will drop the current database and create a new, blank database.

To run a data source, run this command in the MINE_NAME/integrate/ directory, specifying the source name (as it appears in project.xml):

$ ant -v -Dsource=malaria-gff

Most sources have multiple stages in retrieving data, to run just one stage use:

$ ant -v -Dsource=malaria-gff -Daction=[retrieve|load]

The stages are:

retrieve  load data from source database/files into an items database
translate  convert from a source items database to a target items database
load     read from a target items database and integrate into the production database

Most sources do not have a translate step so retrieve will write to the common-tgt-items database.

4 this allows multiple sets of properties file. eg. passing ‘‘-V test‘ causes the build system to look for MINE_NAME.properties.test instead of the default file MINE_NAME.properties
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 testmodel project.

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 `ant build-db` 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. As an example the additions file for uniprot is `bio/sources/uniprot/uniprot_additions.xml`

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 is:

```plaintext
extra.model.paths.start = bio/core/genomic_additions.xml bio/sources/so/so_additions.xml
extra.model.paths.end = bio/core/shortcuts.xml
```

Here `genomic_additions.xml` and `so_additions.xml` will be merged first and `shortcuts.xml` will be merged after all other model fragments.
Note that the `bio/core/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`:

```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`):

```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:

```xml
...<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.

**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 a global file called “MODEL-NAME_keyDefs.properties” which must be in the classpath. Each data source needs an additional configuration file defining which primary keys should be used when integrating data, which can also define some extra keys. These files should be 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: `bio/sources/sourcename/resources/sourcename_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
```

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 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 \_keys.properties files.

Defining keys in a central file

**Note:** This is an older method of defining keys in a central configuration file and just specifying which of those keys should be used in each.

Global primary key configuration file 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
```
Using global keys in each source

In this case the properties file for each data source lists primary key names from the central `genomic_keyDefs.properties` file that can be used when integrating that data source. 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. Each class must be mentioned only once in this file, unlike the global configuration. 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.
Classname = primary_key_name, primary_key_name2
```

This line means that the class “Classname” and all its subclasses have a two primary keys available for this data source, called “primary_key_name” and “primary_key_name2”, which should be defined properly in the global configuration.

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_NAME/postprocess project with:

```bash
$ ant -v
```

To run individual post-process steps use, for example:

```bash
$ ant -v -Daction=create-references
```

When running one postprocess step like this (multiple steps separated by comma is not supported), the action 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 (see above).

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 or modMine 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:

```bash
cd $MINE/dbmodel
ant generate-update-triggers
```

This will generate two SQL files in the build/model/ 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

```bash
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 propagate 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**

```
$INTERMINE/imbuild/integrate.xml:54: The following error occurred while executing this line:  
$INTERMINE/imbuild/source.xml:330: java.lang.RuntimeException: Exception while dataLoad:ng - to allow Problem while loading item identifier 6_1083 because Given a ProxyReference, but id not in ID Map. So  
```

**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.

```sql
select * from item where identifier = '6_1083'
```

This gives us the class and item id:

<table>
<thead>
<tr>
<th>implementations</th>
<th>classname</th>
<th>identifier</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strain</td>
<td>6_1083</td>
<td>1380031</td>
<td></td>
</tr>
</tbody>
</table>

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.

```sql
select * from attribute where itemid = 1380031;
```

This gives us any attributes stored for this object, in our example this gives us the primary identifier.

<table>
<thead>
<tr>
<th>intermine_value</th>
<th>name</th>
<th>itemid</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS2329</td>
<td>primaryIdentifier</td>
<td>1380031</td>
</tr>
</tbody>
</table>

**Data Integrity Checks**

**Template Comparison**

There are template comparison scripts available under *intermine/scripts*, that will run 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 scripts present 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.

Scripts are available in Perl and Python. Installation of the Perl modules requires the installation of various dependencies, so many users may find the Python versions easier to install, since these have fewer dependencies.
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@blackhole.net
```

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.org
```

Perl Script

Dependencies  The Perl script depends on the following modules:

- `Webservice::InterMine` - the InterMine client library.
- `Email::Valid` - Optional email validation.
- `List::Util` - Core perl list extensions. [most systems will have this already]

These can be installed with the following command:

```
$ sudo cpan Webservice::Intermine Email::Valid List::Util
```

Invocation  You can run the script to compare two mines, or you can run the script to simply run all the templates in a mine.

Compare two mines, use one email address for both to and from

```
$ ./compare_templates_for_releases mine-A mine-B email-address
```

Compare one mine to itself, use one email address for both to and from

```
$ ./compare_templates_for_releases mine-A email-address
```

Results  The resulting email will look like this:

```
----------------------------------------------------------------------
-----------------------------------In Both: Diff >= 10%
ChromLocation_CRMOverlappingTFBindingsite release-beta: 42, release-28.0: 213, diff: 80%
```

### Acceptance Tests

#### How to run the tests

1. Add a file to `MINE_NAME/integrate/resources`, eg. `flymine_acceptance_test.conf`

2. run acceptance tests here:

   ```
   $ cd MINE_NAME/integrate
   $ ant acceptance-tests
   ```

   The results will be in `MINE_NAME/integrate/build/acceptance_test.html`

#### Types of tests

You can assert that a query returns true:

```java
assert {
    sql: select count(*) >= 400000 from goannotation
}
```

Or doesn’t have any results:

```java
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:

```sql
some-results {
    sql: select * from organism where name = 'Anopheles gambiae'
    note: We should have an Anopheles gambiae object but not an Anopheles gambiae FEST one
}
```

### 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:

```sql
SELECT classnamea, name, classnameb, count(*)
FROM (SELECT distinct itema.classname AS classnamea, name, itemb.classname AS classnameb, itemb.identifier
      FROM item AS itemA, reference, item
      AS itemB
      WHERE itema.id = itemid AND refid = itemb.identifier)
```
\( \text{AND} \ \text{itema.id} < \text{itemb.id} \) \ 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. \textit{subject} is the name of the reference in Synonym. Changing the store order would improve performance.

<table>
<thead>
<tr>
<th>classnamea</th>
<th>name</th>
<th>classnameb</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonym</td>
<td>subject</td>
<td>Gene</td>
<td>27836</td>
</tr>
</tbody>
</table>

**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:

```properties
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:

```xml
<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 \textit{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 recursively 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() dereferences 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:

<table>
<thead>
<tr>
<th></th>
<th>Load time</th>
<th>objs / min</th>
<th>DB size</th>
<th>tracker size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>4.51 min</td>
<td>1,525,015</td>
<td>9.6 GB</td>
<td>3.7 GB</td>
</tr>
<tr>
<td>No tracker</td>
<td>3.94 min</td>
<td>1,748,446</td>
<td>5.56 GB</td>
<td>1 GB</td>
</tr>
<tr>
<td>Consequence as SimpleObject</td>
<td>3.37 min</td>
<td>2,044,448</td>
<td>4.6 GB</td>
<td>1.4 GB</td>
</tr>
<tr>
<td>Both of above</td>
<td>3.20 min</td>
<td>2,153,291</td>
<td>4.1 GB</td>
<td>1 GB</td>
</tr>
</tbody>
</table>

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...
[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] Reading all employee objects with empty object cache
[run-performance-test] Read 20000 employee objects, took: 272ms.

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

You should expect similar.

Query performance (precomputed tables)

InterMine can make use of precomputed tables (analogous 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.

```
# in <mine>/webapp
$ ant precompute-templates
```

Sometimes it can be slow to precompute complex templates, an ignore flag lets you specify a comma separated list of template names not to precompute. For example:

```
# in <mine>/webapp
$ ant -Dignore=template1,template2 precompute-templates
```

Manual specification of queries

You can specify any IQL query to precompute in the file `<mine>/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.

```
# in <mine>/webapp
> ant precompute-queries
```

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:

```
# in <mine>/webapp
$ ant drop-precomputed-tables
```

### 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, 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 a7_. a2_. a3_. a8_. a5_. a6_ FROM org.intermine.model.bio.Interaction AS a1_, org.intermine.model.bio.Gene AS a2_, org.intermine.model.bio.InteractionDetail AS a3_, org.intermine.model.bio.InteractionExperiment AS a4_, org.intermine.model.bio.InteractionTerm AS a5_, org.intermine.model.bio.Publication AS a6_, org.intermine.model.bio.Gene AS a7_, org.intermine.model.bio.InteractionTerm AS a8_ WHERE (a1_.gene2 CONTAINS a2_ AND a1_.details CONTAINS a3_ AND a3_.experiment CONTAINS a4_ AND a3_.relationshipType CONTAINS a5_ AND a4_.publication CONTAINS a6_ AND a1_.gene1 CONTAINS a7_ AND a4_.interactionDetectionMethods CONTAINS a8_ AND a7_.id IN ? AND a2_.id IN ?) ORDER BY a7_.symbol, a2_.symbol, a3_.name, a3_.role1, a3_.role2, a3_.type, a8_.name, a5_.name, a6_.pubMedId 1: [1007850] 2: [2848406]
```
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_.id AS a5_id, a6_.id AS a6_id, a7_.symbol AS orderbyfield1, a2_.symbol AS orderbyfield2, a3_.name AS orderbyfield3, a3_.role1 AS orderbyfield4, a3_.role2 AS orderbyfield5, a3_.type AS orderbyfield6, a8_.name AS orderbyfield7, a5_.name AS orderbyfield8, a6_.pubMedId AS orderbyfield9 FROM Interaction AS a1_, Gene AS a2_, InteractionDetail AS a3_, InteractionExperiment AS a4_, InteractionTerm AS a5_, Publication AS a6_, Gene AS a7_, InteractionTerm AS a8_, InteractionDetectionMethodsInteractionExperiment AS indirect0 WHERE a1_.gene2Id = a2_.id AND a1_.id = a3_.interactionId AND a3_.experimentId = a4_.id AND a3_.relationshipTypeId = a5_.id AND a4_.publicationId = a6_.id AND a1_.gene1Id = a7_.id AND a4_.id = indirect0.interactionExperiment AND indirect0.interactionDetectionMethods = a8_.id AND a7_.id IN (1007850) AND a2_.id IN (2848406) ORDER BY a7_.symbol, a2_.symbol, a3_.name, a3_.role1, a3_.role2, a3_.type, a8_.name, a5_.name, a6_.pubMedId, a7_.id, a2_.id, a3_.id, a8_.id, a5_.id, a6_.id LIMIT 5000

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_id, a5_.id AS a5_id, a6_.id AS a6_id, P98.a1_symbol AS orderbyfield1, P98.a3_symbol AS orderbyfield2, P96.name AS orderbyfield3, P96.role1 AS orderbyfield4, P96.role2 AS orderbyfield5, P96.type AS orderbyfield6, a8_.name AS orderbyfield7, a5_.name AS orderbyfield8, a6_.pubMedId AS orderbyfield9 FROM precomp_45503 AS P98, InteractionDetail AS P96, InteractionExperiment AS P97, InteractionTerm AS a5_, Publication AS a6_, InteractionTerm AS a8_, InteractionDetectionMethodsInteractionExperiment AS indirect0 WHERE P98.a2_id = P96.interactionId AND P96.experimentId = P97.id AND P96.relationshipTypeId = a5_.id AND P97.publicationId = P97.id AND indirect0.interactionExperiment AND indirect0.interactionDetectionMethods = a8_.id AND P98.a1_id IN (1007850) AND P98.a3_id IN (2848406) ORDER BY P98.a1_symbol, P98.a3_symbol, P96.name, P96.role1, P96.role2, P96.type, a8_.name, a5_.name, a6_.pubMedId, P98.a1_id, P98.a3_id, P96.id, a8_.id, a5_.id, a6_.id LIMIT 5000

bag tables: 0 ms, generate: 0 ms, optimise: 0 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:

```
$ ant run-iql-query -Dquery='some IQL'
```

**Useful ObjectStore properties**

**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.
Guide to Customising your Web Application

Guide to Customising Blue Genes

Certain features of the blue genes app are controlled by parameters set in the web.properties file.

<table>
<thead>
<tr>
<th>purpose</th>
<th>parameters</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>default examples for the ID resolver</td>
<td>bag.example.identifiers.protein=Q8T3M3,FBpp0081318,FT2Z_DROME and bag.example.identifiers=CG9151, FBgn0000099 (one per type)</td>
<td></td>
</tr>
<tr>
<td>default separators</td>
<td>list.upload.delimiters</td>
<td></td>
</tr>
<tr>
<td>default regionsearch</td>
<td>genomicRegionSearch.*</td>
<td>H. sapiens (note: please do not use long format, e.g. Homo sapiens)</td>
</tr>
<tr>
<td>default query builder query</td>
<td>&lt;query model=&quot;genomic&quot; view=&quot;Gene.secondaryIdentifier Gene.symbol Gene.primaryIdentifier Gene.organism.name&quot; sortOrder=&quot;Gene.secondaryIdentifier ASC&quot; &gt;&lt;constraint path=&quot;Gene.organism.name&quot; op=&quot;=&quot; value=&quot;Homo sapiens&quot; code=&quot;A&quot; /&gt;&lt;/query&gt;</td>
<td>e.g. PPARG, Insulin, rs876498</td>
</tr>
</tbody>
</table>

Please see *Features* for details on these parameters.

**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:
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 `<your_mine_name>/webapp/resources/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</strong>` 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:

```plaintext
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:

```plaintext
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:
Example:

```bash
# 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:

```bash
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):
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):

```plaintext
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:

```plaintext
links.blog = http://<your_blog>
```

**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:

```plaintext
<class className="org.intermine.model.bio.BioEntity">
    <headerconfig>
        <titles>
            <title mainTitles="symbol|primaryIdentifier" numberOfMainTitlesToShow="1" subTitles="*organism.shortName*"/>
        </titles>
    </headerconfig>
</class>
```
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.

```xml
<headerconfig>
  <customlinks>
    <customlink
      url="http://flybase.org/reports/{primaryIdentifier}.html"
      image="flybase_logo_link.png"
    />
  </customlinks>
</headerconfig>
```

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.: flymine/webapp/resources/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 overridden 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:

<table>
<thead>
<tr>
<th>14 dataSets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>url</td>
</tr>
<tr>
<td>FlyBase data set for Drosophila melanogaster</td>
<td></td>
</tr>
<tr>
<td>SwissProt data set</td>
<td><a href="http://ca.expasy.org/sprot">http://ca.expasy.org/sprot</a></td>
</tr>
<tr>
<td>Drosophila 12 Genomes Consortium terminology</td>
<td></td>
</tr>
<tr>
<td>KEGG orthologues data set</td>
<td></td>
</tr>
<tr>
<td>TreeFam data set</td>
<td><a href="http://www.treefam.org/cgi-bin/miles_page.pl?download">http://www.treefam.org/cgi-bin/miles_page.pl?download</a></td>
</tr>
<tr>
<td>GO Annotation from FlyBase</td>
<td></td>
</tr>
<tr>
<td>REDfly Drosophila transcription factor binding sites</td>
<td><a href="http://redfly.ccr.buffalo.edu/?content=--search.php">http://redfly.ccr.buffalo.edu/?content=--search.php</a></td>
</tr>
<tr>
<td>REDfly Drosophila transcriptional cis-regulatory modules</td>
<td><a href="http://redfly.ccr.buffalo.edu/?content=--search.php">http://redfly.ccr.buffalo.edu/?content=--search.php</a></td>
</tr>
<tr>
<td>DRSC data set</td>
<td><a href="http://flymail.org/">http://flymail.org/</a></td>
</tr>
<tr>
<td>FlyBase fasta data set for Drosophila melanogaster</td>
<td></td>
</tr>
</tbody>
</table>

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:
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 within 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 overridden 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**

**Warning:** Report Displayers can only be embedded on a report page and not elsewhere, furthermore they are tightly knit to the Java system. Check out *Report Widgets* for a more flexible arrangement.

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 display 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 trunk/bio/webapp/src/org/intermine/bio/web/displayers/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 trunk/flymine/webapp/resources/webapp/WEB-INF/webconfig-model.xml.

For examples of the common displayers and configuration details please see ReportDisplayerGallery.

**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.

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 $<mine>/webapp/resources/webapp/model$ to display the results.
4. Add configuration to $<mine>/webapp/resources/webapp/WEB-INF/webconfig-model.xml$ 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:

```java
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).

---

5 ReportDisplayer makes available a variable called **im** which is the **InterMineAPI** which provides access to config and query execution classes.
Protein sequence

Applicable for Protein - shows length, sequence export.

GBrowse

Show an inline image from a configured GBrowse instance.

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

Homologues

Shows a table of organism and homologous genes of homologues per organism.
Figure 1.10: A Genome browser view in FlyMine.

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.

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.
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.

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.
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 its own exons.

Complexes - Protein interactions

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
- S1AcceptorSite: 6
- TFBindingSite: 12
- TSS: 4
- ThreePrimeUTR: 6
- Transcript: 354
- TranscriptRegion: 543
- TranscriptionEndSite: 15

Figure 1.16: An Overlapping features displayer in modMine.

```xml
<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.
InterMine Table Colors Conversion Table

<table>
<thead>
<tr>
<th>Color</th>
<th>Hex</th>
<th>Hex</th>
<th>Hex</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>#002575</td>
<td>#e2f5fe</td>
<td>#eef0f7</td>
<td>#fbfdfe</td>
</tr>
<tr>
<td>bright_blue</td>
<td>#002575</td>
<td>#df00dd</td>
<td>#e6f0e4</td>
<td>#f8f7f7</td>
</tr>
<tr>
<td>brown</td>
<td>#f8f7f7</td>
<td>#f00000</td>
<td>#bebebe</td>
<td>#fafa2b</td>
</tr>
<tr>
<td>ecoli_blue</td>
<td>#004499</td>
<td>#f00000</td>
<td>#bebebe</td>
<td>#fafa2b</td>
</tr>
<tr>
<td>gold</td>
<td>#662f2e</td>
<td>#f75fda</td>
<td>#f7f1ee</td>
<td>#fefe06</td>
</tr>
<tr>
<td>green</td>
<td>#5e664c</td>
<td>#dbe0b7</td>
<td>#f0f0f0</td>
<td>#f9ff00</td>
</tr>
</tbody>
</table>
Figure 1.19: FlyAtlas gene expression data in FlyMine, this uses the Google Data Visualization 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 values are derived from RNA-seq data from the Chen and Li groups and are log2 of the actual value.

Heatmaps visualization powered by canvasXpress, learn more about the display options.

Click to see/hide the expression maps

Figure 1.21: Heatmap of fly gene expression in modMine, this makes use of canvasXpress JavaScript library.

Inline List

```html
<div class="inline-list">
  <h3>2 probeSets</h3>
  <ul>
    <li>FBgn0014159</li>
    <li><a>Complementation group F</a></li>
  </ul>
</div>
```

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div.inline-list</td>
<td>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</td>
</tr>
<tr>
<td>h3 (optional)</td>
<td>header 3 (see below) styling</td>
</tr>
<tr>
<td>ul</td>
<td>list we be displayed inline, without margins between items and without list styles (circles, squares etc.)</td>
</tr>
</tbody>
</table>

Inline List (Inactive, No Results)

```html
<div class="inline-list gray">
  <h3>0 probeSets</h3>
</div>
```

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div.inline-list.gray</td>
<td>one can apply an ‘inactive’ theme by attaching a class to the top element</td>
</tr>
</tbody>
</table>
Inline List (Tagging, Right)

```html
<div class="inline-list">
  <h3><div class="right">Right positioned</div> 0 probeSets</h3>
</div>
```

CSS Description

<table>
<thead>
<tr>
<th>div.inline-list</th>
<th>div.right (optional)</th>
<th>will float element to the right and apply appropriate colors to links; needs to go first, before any other text</th>
</tr>
</thead>
</table>

‘Header’ Inline List

```html
<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

<table>
<thead>
<tr>
<th>div.inline-list .name (optional)</th>
<th>the main theme color will be applied to the element</th>
</tr>
</thead>
</table>

Collection Table

```html
2 clones

<table>
<thead>
<tr>
<th>secondaryIdentifier</th>
<th>primaryIdentifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT01163</td>
<td>Fbc00383804</td>
</tr>
<tr>
<td>RT01063</td>
<td>Fbc00383805</td>
</tr>
</tbody>
</table>
```

```html
<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>
```
InterMine Documentation, Release

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div.collection-table h3</td>
<td>table title will pickup theme colors much like Title (Level 3) below</td>
</tr>
<tr>
<td>div.collection-table thead</td>
<td>table expects a thead element, that will apply the same background</td>
</tr>
<tr>
<td>th,td</td>
<td>as the title</td>
</tr>
<tr>
<td>div.collection-table.nowrap</td>
<td>row columns do not wrap and are displayed inline</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Comment</th>
<th>Proteins</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>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 segment polarity genes.</td>
<td>EVE_DROME</td>
</tr>
<tr>
<td>similarly</td>
<td>Belongs to the even-skipped homeobox family.</td>
<td>EVE_DROME</td>
</tr>
<tr>
<td>similarly</td>
<td>Contains 1 homeobox DNA-binding domain.</td>
<td>EVE_DROME</td>
</tr>
</tbody>
</table>

**Collection Table (Type Column, Text Highlight)**

```html
<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>
```

---

**Collection Table (Vertical Column Border)**

```html
<div class="collection-table column-border">
  <table>
    -- ... -->
  </table>
</div>
```

---

**Genome feature**

<table>
<thead>
<tr>
<th>Sequence ontology type:</th>
<th>length:</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene</td>
<td>1539 Fasta...</td>
</tr>
<tr>
<td>Location:</td>
<td>Cyto location:</td>
</tr>
<tr>
<td>2R:5866746-5868284</td>
<td>46C10-46C10</td>
</tr>
</tbody>
</table>

---

Chapter 1. Contents
InterMine Documentation, Release

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div.collection-table.column-border</td>
<td>uses a pseudoclass to apply a border between columns</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Sequence ontology type</th>
<th>Length</th>
<th>Cyto location</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene</td>
<td>1539</td>
<td>46C10-46C10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2R:5866746-5868284 forward strand</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Collection Table (Vertical Column Border by 2)**

```html
<div class="collection-table column-border-by-2">
  <table>
  </table>
</div>
```

**CSS**

<table>
<thead>
<tr>
<th>div.collection-table.column-border-by-2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uses a pseudoclass to apply a border between every other column</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Genome feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence ontology type: gene</td>
</tr>
<tr>
<td>Length: 1539</td>
</tr>
<tr>
<td>Cyto location: 46C10-46C10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>2R:5866746-5868284</td>
</tr>
</tbody>
</table>

**Collection Table (Inactive, No Results)**

```html
<div class="collection-table gray">
  <h3>0 genes</h3>
</div>
```

**CSS**

<table>
<thead>
<tr>
<th>div.collection-table.gray</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>one can apply an ‘inactive’ theme by attaching a class to the top element</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 miRNA targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 miRNA targets</td>
</tr>
</tbody>
</table>

**Collection Table (Tagging, Right)**

```html
<div class="collection-table">
  <h3 class="right">Right positioned</h3>
  0 genes</div>
```

**Collection Table (Tagging, Right)**

```html
<div class="collection-table">
  <h3><div class="right">Right positioned</div> 0 genes</h3>
</div>
```
<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div.collection-table</td>
<td>will float element to the right and apply appropriate colors to links; needs to go first, before any other text</td>
</tr>
</tbody>
</table>

### Collection Table (Persistent Table Headers)

```html
div.collection-table
  <-- ... -->
</div>
```

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div.collection-table.persistent</td>
<td>will make table headers persist as you scroll within the table</td>
</tr>
</tbody>
</table>

### Basic Table (Generic)

```html
div
  h3>Some title</h3>
  <table>
    <tr><td>Row column</td></tr>
  </table>
</div>
```

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div.basic-table h3</td>
<td>will apply the heading 3 style (see below)</td>
</tr>
<tr>
<td>div.basic-table</td>
<td>will float element to the right and apply appropriate colors to links; needs to go first, before any other text</td>
</tr>
<tr>
<td>div.right (optional)</td>
<td>needs to go first, before any other text</td>
</tr>
<tr>
<td>div.basic-table table</td>
<td>will make sure that the table is properly collapsed, has padding and does not have cellspacing</td>
</tr>
<tr>
<td>div.basic-table.gray (optional)</td>
<td>one can apply an ‘inactive’ theme by attaching a class to the top element</td>
</tr>
</tbody>
</table>
Regulatory Regions

CRM

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Chromosome Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>eve_late_element_2_even7</td>
<td>2R: 5868446-5870246</td>
</tr>
<tr>
<td>eve_proximal_promoter_inc_TATA</td>
<td>2R: 5866610-5866826</td>
</tr>
<tr>
<td>eve_MHE</td>
<td>2R: 5872788-5873100</td>
</tr>
<tr>
<td>eve_EME-B</td>
<td>2R: 5872666-5873261</td>
</tr>
<tr>
<td>eve_EME-B3</td>
<td>2R: 5872999-5873261</td>
</tr>
<tr>
<td>eve_EME-B5</td>
<td>2R: 5872666-5873099</td>
</tr>
<tr>
<td>eve_eme2</td>
<td>2R: 5872939-5873239</td>
</tr>
<tr>
<td>eve_mas</td>
<td>2R: 5861443-5861548</td>
</tr>
<tr>
<td>eve_stripe_3.7</td>
<td>2R: 5863005-5863516</td>
</tr>
<tr>
<td>eve_early_APR</td>
<td>2R: 5869447-5870352</td>
</tr>
</tbody>
</table>

Collection of Collection Tables

```html
<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>
```

CSS

```
div.collection-of-collections
  a {text-decoration: underline;}
div.collection-of-collections a.active
  a {text-decoration: none; font-weight: bold;}
```

Table Togglers (Less, More, Expand, Collapse, Show in table)

```html
<div class="collection-table">
  <!-- ... -->
  <div class="toggle">~ Show more rows</div>
</div>
```

1.6. Guide to Customising your Web Application
InterMine Documentation, Release

CSS | Description
---|---
div.collection-table | will create apply an expand/more button
div.toggle a more | 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.toggle a less | a generic button without any upward/downward arrows
div.collection-table div.toggle a (optional) 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

CSS | Description
---|---
h3.goog | will pickup theme colors and apply Report Page/Google News-style colors, backgrounds, borders

Smallfont, Display one-per-line

<table>
<thead>
<tr>
<th>Homologues</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. gambiae</td>
</tr>
<tr>
<td>AGAP010279</td>
</tr>
<tr>
<td>CG30401</td>
</tr>
</tbody>
</table>

CSS | Description
---|---
.tiny-font | will apply 11px font size to all descendants

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)

```html
<div class="loading-spinner"></div>
```

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.loading-spinner</code></td>
<td>will show an inline block positioned loading spinner gif</td>
</tr>
</tbody>
</table>

Table Warning

![Pathways from Other Mines](image)

There is a problem with this displayer.

```html
<div class="collection-table warning">
  <-- ... -->
</div>
```

<table>
<thead>
<tr>
<th>CSS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.warning</code></td>
<td>will show a warning icon and change the color of the div to pale yellow</td>
</tr>
</tbody>
</table>

Report Widgets

Have been retired and made into something better. Use Report Displayers instead.

---

See Apps/A or Apps/C Grunt Build. The changes are as follows:

1. Widget becomes an App.
2. The main class in `presenter.[js|ts|ls|coffee]` has been changed from `Widget` to `App`.
3. A config file with dependencies for a particular App is moved to its folder in the form of a `config.js` file. The contents of this file is being exported and contains example config to be passed from middleware and/or client.
4. The Java service serving Report Widgets has been retired from InterMine core.
5. The Node.js service now works as a middleware. You can plug it into your (Node.js) app by passing an array of paths on local or remote systems where it can find the App sources.
6. There is a repo for the Middleware and for the Sources.
7. Folders and modules are now allowed and can be required across.
8. The root `App` now needs to be exported from the main class.
9. To use a template you need to include its suffix.

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 the 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:

```sql
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:

```xml
<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  `osbag_int` table

<table>
<thead>
<tr>
<th><strong>column</strong></th>
<th><strong>notes</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>bagid value</td>
<td>unique bag id</td>
</tr>
<tr>
<td></td>
<td>intermine object id</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th><strong>column</strong></th>
<th><strong>notes</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>osbid type id</td>
<td>bag id</td>
</tr>
<tr>
<td>name</td>
<td>type of object, eg. Gene</td>
</tr>
<tr>
<td>datecreated</td>
<td>id</td>
</tr>
<tr>
<td>description</td>
<td>name of list, provided by user</td>
</tr>
<tr>
<td>userprofileid</td>
<td>timestamp</td>
</tr>
<tr>
<td>intermine_state</td>
<td>description, provided by user</td>
</tr>
<tr>
<td></td>
<td>user id</td>
</tr>
<tr>
<td></td>
<td>CURRENT, NOT_CURRENT or TO_UPGRADE</td>
</tr>
</tbody>
</table>

**bagvalues table**
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 regenerated 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 upgraded; 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.

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.

```html
<!-- 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.

```javascript
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.

```javascript
// 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.

```javascript
// 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.

```javascript
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/', token);
// 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 occasions 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:

```javascript
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:

```javascript
var options = {
  resultsCb: function(pq) {
    // ...
  }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Clicking on Create list button in a popover window:

```javascript
var options = {
  listCb: function(pq) {
    // ...
  }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

I want to hide the title or description of a widget.

```javascript
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:

```javascript
// 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:

\[
P = \frac{\binom{M}{k} \binom{N-M}{n-k}}{\binom{N}{n}}
\]

**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

Note: The relevant Java source.

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 = \( p\text{-value} \times (\text{number of tests} - \text{rank}) \)

**Benjamini Hochberg**  This correction is the less stringent than the Bonferroni, and therefore tolerates more false positives.

Adjusted p-value = \( p\text{-value} \times (\text{number of tests/rank}) \)

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 \( P = P \times CCGO \)

where the correction coefficient CCGO is calculated as:

\[
CCGO = \frac{LGO/LWH}{NGO/NWG}
\]

**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.
PubMed id: 15297299

Controlling the false discovery rate: a practical and powerful approach to multiple testing
Benjamini, Yoav; Hochberg, Yosef

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.

Variable locus length in the human genome leads to ascertainment bias in functional inference for non-coding elements

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.

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique id used by javascript only. Spaces not allowed.</td>
<td>unique_id</td>
</tr>
<tr>
<td>pathString</td>
<td>which collection to use in the widget</td>
<td>Gene.homologues[type=orthologue].homologue.organism</td>
</tr>
<tr>
<td>exportField</td>
<td>which field from the objects in your list to export</td>
<td>primaryIdentifier</td>
</tr>
<tr>
<td>typeClass</td>
<td>types of lists that should display this widget. Use the simple class name</td>
<td>Gene</td>
</tr>
</tbody>
</table>

The following are optional attributes:

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>appears at the top of the widget</td>
<td>Orthologues</td>
</tr>
<tr>
<td>description</td>
<td>description of the widget</td>
<td>Counts of orthologues</td>
</tr>
<tr>
<td>displayField</td>
<td>which fields from the objects in the collection (in the above example, Gene.proteins) to display, eg. primaryAccession</td>
<td>name</td>
</tr>
<tr>
<td>columnTitle</td>
<td>heading for the “count” column</td>
<td>Orthologues</td>
</tr>
<tr>
<td>externallink</td>
<td>link displayed next to first column, identifier will be appended to link</td>
<td>symbol</td>
</tr>
<tr>
<td>externallinkLabel</td>
<td>label for external link</td>
<td></td>
</tr>
<tr>
<td>views</td>
<td>path fields display in the query running when the user clicks on the widget</td>
<td></td>
</tr>
</tbody>
</table>

**Graph/Chart widgets**  Graph widgets display datasets in graphical format.

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique id used by javascript only. Spaces not allowed.</td>
<td>unique_id</td>
</tr>
<tr>
<td>graphType</td>
<td>which type of chart to render</td>
<td>ColumnChart,&quot;BarChart&quot;, PieChart</td>
</tr>
<tr>
<td>start-Class</td>
<td>it’s the root class for all the paths specified in the configuration.</td>
<td>Gene</td>
</tr>
<tr>
<td>type-Class</td>
<td>type of lists that should display this widget. Use the simple class name.</td>
<td>Gene</td>
</tr>
<tr>
<td>category-Path</td>
<td>Must be attribute. We can specify the subclass using the syntax path[subclass type]</td>
<td>mRNAExpressionResults.stageRange, mRNAExpressionResults.expressed</td>
</tr>
<tr>
<td>series-Path</td>
<td>the series path. This has to be an attribute. We can specify the subclass using the syntax path[subclass type]</td>
<td>mRNAExpressionResults.expressed, mRNAExpressionResults.expressed</td>
</tr>
<tr>
<td>series-Values</td>
<td>the values of different series. Case sensitive. You can specify boolean values</td>
<td>true, false or Up, Down</td>
</tr>
<tr>
<td>series-Labels</td>
<td>the labels displayed on the graphs to distinguish inside a category the different series</td>
<td>Expressed, Not Expressed or Up, Down</td>
</tr>
<tr>
<td>views</td>
<td>attributes paths displayed when the user clicks an area on the graph</td>
<td>name, organism.name</td>
</tr>
</tbody>
</table>


---

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
### 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

<table>
<thead>
<tr>
<th>Organism.name</th>
<th>Orthologues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homo sapiens</td>
<td>60</td>
</tr>
<tr>
<td>Mus musculus</td>
<td>53</td>
</tr>
<tr>
<td>Rattus norvegicus</td>
<td>53</td>
</tr>
<tr>
<td>Danio rerio</td>
<td>41</td>
</tr>
<tr>
<td>Drosophila pseudoobscura</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila simulans</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila erecta</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila virilis</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila yakuba</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila sechellia</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila persimilis</td>
<td>8</td>
</tr>
</tbody>
</table>
Figure 1.22: An example chart widget of BDGP Expression Patterns in FlyMine.
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:

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>appears at the top of the widget</td>
<td>BDGP expression patterns</td>
</tr>
<tr>
<td>description</td>
<td>description of the widget</td>
<td>Expression patterns</td>
</tr>
<tr>
<td>domain</td>
<td>Label displayed on x-axis in the ColumnChart (on y-axis in the BarChart)</td>
<td>Stage</td>
</tr>
<tr>
<td>range</td>
<td>Label displayed on y-axis in the ColumnChart (on x-axis in the a BarChart)</td>
<td>Gene count</td>
</tr>
<tr>
<td>filters</td>
<td>label for filter form field</td>
<td>Organism</td>
</tr>
<tr>
<td>filter</td>
<td>the values for the filter, set in the dropdown</td>
<td>All,KEGG pathways,Reactome data</td>
</tr>
<tr>
<td>listPath</td>
<td>the path used to build the bag constraint</td>
<td>FlyAtlasResult.material</td>
</tr>
<tr>
<td>constraints</td>
<td>separated by comma, case sensitive, must be attributes, operator can be = or /=</td>
<td>organism.name=[Organism]</td>
</tr>
</tbody>
</table>

Note: The graphs use Google Visualization 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.

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique id used by JavaScript only. Spaces not allowed.</td>
<td>unique_id</td>
</tr>
<tr>
<td>startClass</td>
<td>Root class for all the paths specified in the configuration. Use simple name (e.g. Gene)</td>
<td>Gene</td>
</tr>
<tr>
<td>startClassDisplay</td>
<td>Field displayed when user clicks on the widget on 'Matches' column</td>
<td>primaryIdentifier</td>
</tr>
<tr>
<td>typeClass</td>
<td>Type of lists that should display this widget. Use the simple class name.</td>
<td>Gene</td>
</tr>
<tr>
<td>enrich</td>
<td>Field to be enriched, displayed in the widget in the firts column</td>
<td>goAnnotation.ontologyTerm.parents.name</td>
</tr>
<tr>
<td>views</td>
<td>attributes paths displayed when the user clicks on View results button</td>
<td>symbol,organism.name</td>
</tr>
</tbody>
</table>

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:

root class associated to the category path.

7We 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.

8Optional if the startClass contains the bag type class.

9For the values we can use static values or the selected filter value using the syntax: path constraint = [filter identifier].

10organism’s name matching with the value selected in the filter with filterLabel ‘Organism’

11You 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.

<table>
<thead>
<tr>
<th>Test Correction</th>
<th>Max p-value</th>
<th>Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holm-Bonferroni</td>
<td>0.05</td>
<td>biological_process</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GO Term</th>
<th>p-Value</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>regulation of transcription, DNA-dependent [Link]</td>
<td>8.613623e-22</td>
<td>23</td>
</tr>
<tr>
<td>regulation of RNA biosynthetic process [Link]</td>
<td>8.613623e-22</td>
<td>23</td>
</tr>
<tr>
<td>transcription, DNA-dependent [Link]</td>
<td>6.709974e-21</td>
<td>23</td>
</tr>
<tr>
<td>RNA biosynthetic process [Link]</td>
<td>7.113099e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of RNA metabolic process [Link]</td>
<td>7.113099e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of macromolecule biosynthetic process [Link]</td>
<td>9.236209e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of cellular macromolecule biosynthetic process [Link]</td>
<td>9.236209e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of transcription from RNA polymerase II promoter [Link]</td>
<td>1.271350e-20</td>
<td>19</td>
</tr>
</tbody>
</table>

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 google
**externalLink** | link displayed next to first column | organism.name=[list] Ontology
**filters** | extra filters to add to the display | organism.name=[list] google
**filterLabel** | label for filter form field | goAnnotation.ontologyTerm.identifier
**enrichIdentifier** | identifier for the row displayed, if not specified, enrich field used | results.expressed = true
**constraints** | constraints separated by comma. The paths have to be attributes. The operator can be = or != | organism.name=[list] goAnnotation.ontologyTerm.identifier
**constrainsForView** | constraints separated by comma used for building the query executed when the user clicks on the widget on ‘Matches’ column | set for org.intermine.bio.web.widget.GeneLenghtCorrectionCoefficient to normalize by gene length
**correctionCoefficient** | set to org.intermine.bio.web.widget.GeneLenghtCorrectionCoefficient to normalize by gene length |

**Examples**  See other mines’ config files for more examples, eg:

- branches/intermod_workshop/flymine/webapp/resources/webapp/WEB-INF/webconfig-model.xml in FlyMine
- branches/intermod_workshop/modmine/webapp/resources/webapp/WEB-INF/webconfig-model.xml in modMine
- branches/intermod_workshop/metabolicmine/webapp/resources/webapp/WEB-INF/webconfig-model.xml in metabolicMine

**Background population**  In the enrichment 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 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.

---

12Use 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.

13Specify only one. This has to be an attribute. Used in the results table. Specify the subclass using the syntax path[subclass type].

14Case 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.
Displaying widgets

JavaScript

Widget service  Create a new Widgets instance pointing to a service:

```javascript
var widgets = new intermine.widgets("http://beta.flymine.org/query/service/");
```

Choose a widget  Choose which widget(s) you want to load:

```javascript
// 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  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.

Note: Online example can be visited at tinkerbin.

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:

   ```html
   <script src="http://cdn.intermine.org/api"></script>
   ```

3. Load the Widget Service:

   ```javascript
   <script type="text/javascript">
   intermine.load('widgets', function() {
     var Widgets = new intermine.widgets('http://beta.flymine.org/query/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/query/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">`.

7. Add HTML, eg:

```html
<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/query/service/');
            // Load all Widgets:
            Widgets.all('Gene', 'myList', '#all-widgets');
        });
    </script>
</head>
<body>
    <!-- DIV goes here -->
    <div class="widget" id="all-widgets">
    </div>
</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.

**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.

   ```xml
   <post-processing>
   ...
   <post-process name="create-autocomplete-index"/>
   </post-processing>
   ```

2. In the /postprocess directory, run this command:

   ```
   # in malariamine/postprocess
   $ ant -Daction=create-autocomplete-index
   ```

   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 default
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.

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:

```javascript
$('#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 /webapp/properties/javascript-properties.

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 unnecessary 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:


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’:

```java
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:

```java
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:

```java
intermine.scope('intermine.options.ExternalLinkIcons', {
});
```

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.

**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](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 `classDeccriptions.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-properties` 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:

```xml
<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:

```xml
<class className="org.intermine.model.testmodel.Manager">
  <fields>
    <fieldconfig fieldExpr="address" showInQB="false"/>
  </fields>
</class>
```

**Keyword Search**

Indexing the database runs as a post-process step which creates the index in a directory. The index is then zipped and stored in the database, when you deploy a webapp pointing at the database it will extract the index again. For FlyMine indexing takes less than an hour, including a large proportion of the database.

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 view entire the entire index: Navigate to search results page without search parameter, eg http://www.flymine.org/query/keywordSearchResults.do

**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

## 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 `<MINE>/postprocess`:

```
$ ant -Daction=create-search-index
```

You need to re-release your webapp. Take a look in the intermine.log file when it’s done to see which tables and fields were indexed.

```
2013-02-04 09:26:35 INFO org.intermine.web.search.KeywordSearch - Indexing - Special References:
2013-02-04 09:26:35 INFO org.intermine.web.search.KeywordSearch - Indexing - Facets:
2013-02-04 09:26:35 INFO org.intermine.web.search.KeywordSearch - - field = Category, name = Category, type = SINGLE
2013-02-04 09:26:35 INFO org.intermine.web.search.KeywordSearch - - field = organism.shortName, name = Organism, type = SINGLE
2013-02-04 09:26:35 INFO org.intermine.web.search.KeywordSearch - Indexing with and without attribute prefixes:
2013-02-04 09:26:35 INFO org.intermine.web.search.KeywordSearch - Starting fetcher thread...
2013-02-04 09:26:35 INFO org.intermine.web.search.KeywordSearch - Preparing indexer...
```

You can view the index to see what’s actually in it.

1. Prevent the code from deleting the index once it’s finished. You can do this by commenting out this line:
   ```java
   https://github.com/intermine/intermine/blob/dev/intermine/web/main/src/org/intermine/web/search/KeywordSearch.java#L1906
   ```
   (be sure to put this back!)

2. Use luke: [http://www.getopt.org/luke](http://www.getopt.org/luke) or similar to view the records in the index.
Lucene

Our search uses Lucene’s whitespace analyser, which only uses whitespace to mark word boundaries.

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/resources/webapp/model/images webapp/resources/webapp/model/images/logo.png. 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/InterMineWebApp.properties. 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/resources/model.properties. Properties set in this file overwrite the ones set in intermine/webapp/main/resources/webapp/WEB-INF/classes/InterMineWebApp.properties. Below is an example of how to add tabs to your mine. Replace “api” with the name of your new tab.

First, add your tab to intermine/webapp/main/resources/webapp/headMenu.jsp

```html
<li id="api" test="$tab == 'api'">class="activelink">
    <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/resources/model.properties file:

```properties
# HEADER
menu.api = API
```

You’ll need to configure our web framework (Struts) to properly load your JSP page:

```xml
# in MINE_NAME/webapp/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/resources/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:

```html
<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/resources/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 `intermine/webapp/main/resources/webapp/themes/*`. Explore the folder to see the themes available.

To switch a theme edit `Features`:

```properties
# 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 `ant-clean` to ensure that all of the old files are deleted:

```bash
# in <your_mine>/webapp/resources/webapp
$ ant clean
$ ant default remove-webapp release-webapp
```
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 intermine/webapp/main/resources/webapp/themes/, copy the contents of another theme directory into it and start editing.

Region Search

Bioseg Installation

You must have Bioseg installed or be using Postgres 9.2+ to use the region search.

Configuration

struts-config-model.xml

<definition name="genomicRegionSearchOptions.page" extends="layout.template">
  <put name="body" value="genomicRegionSearchOptions.tile" />
  <put name="pageTitle" value="genomicRegionSearch" />
</definition>
<definition name="genomicRegionSearchOptions.tile" path="/model/genomicRegionSearchOptionsBase.jsp" controllerUrl="/initGenomicRegionSearchOptions.do" />

tiles-defs-model.xml

<definition name="genomicRegionSearchResults.page" extends="layout.template">
  <put name="body" value="/model/genomicRegionSearchResultsBase.jsp" />
  <put name="pageTitle" value="genomicRegionSearchResults" />
</definition>
InterMine Documentation, Release

---

 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 upload
 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.allRegionInvalid = All regions are invalid. Please do a new search.
 genomicRegionSearch.organismEmpty = Organism is empty, please check the data is loaded.

 genomicsRegionSearchOptions.tab = genomicRegionSearch
 genomicsRegionSearchOptions.title = Genomic Regions Search
 genomicsRegionSearchResults.tab = genomicRegionSearch
 genomicsRegionSearchResults.title = Genomic Regions Search Results

 menu.genomicRegionSearch = Regions
 menu.genomicRegionSearchOptions = Genomic Region Search
 menu.genomicRegionSearchResults = Genomic Region Search Results

 web.properties

genomicsRegionSearch.display = true
 genomicsRegionSearch.service =
genomicsRegionSearch.optionsJavascript =
genomicsRegionSearch.resultsJavascript =
genomicsRegionSearch.optionsCss =
genomicsRegionSearch.resultsCss =

## Make sure pre-defined organisms have chromosome location information in the database
 genomicsRegionSearch.defaultOrganisms = D. melanogaster

## Exclude feature types for all organisms, comma separated
 genomicsRegionSearch.featureTypesExcluded.global = GeneFlankingRegion,YouNameItClass

## Exclude feature types for specified organism, semi-colon separated
 genomicsRegionSearch.featureTypesExcluded.byOrganism = D. melanogaster:GeneFlankingRegion,YouNameItClass;
 genomicsRegionSearch.defaultSpans = 2L:14615455..14619002\n2R:5866646..5868384\n3R:2578486..2580016
 genomicsRegionSearch.caption = Search for features that overlap a list of genome coordinates you enter, e.g. <b>2L:11334..12296</b>
 genomicsRegionSearch.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>
     <li><b>chromosome:start-end</b>, e.g. <i>2R:5866746-5868284</i></li>
     <li>tab delimited</li>
 </ul>
 <li>Both <b>base coordinate</b> (e.g. BLAST, GFF/GFF3) and <b>interbase</b> are supported</li>
 <li>Each genome region needs to take a <b>new line</b></li>
 </ul>

## Query fields when export results as csv/tsv
 genomicsRegionSearch.query.Gene.views = {0}.primaryIdentifier,{0}.symbol,{0}.chromosomeLocation.locatedOn.primaryIdentifier,{0}.chromosomeLocation.start,{0}.chromosomeLocation.end,{0}.organism.shortName
 genomicsRegionSearch.query.Gene.sortOrder = {0}.chromosomeLocation.start asc

## 10,000 is the default value, only set if you want a different value
 genomicsRegionSearch.initBatchSize = 10000

• Update defaultOrganisms property as needed
• to disable genomic region search, set genomicsRegionSearch.display = false
• also add genomicsRegionSearch to layout.fixed, e.g.
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 webservies + html

Update IQL query to pathquery  Currently, region query is constructed by IQL (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 webservice, the output will be a list of results tables or a single results table grouped by region.

Ref https://github.com/intermine/intermine/blob/dev/bio/webapp/src/org/intermine/bio/web/logic/GenomicRegionSearchUtil.java#L270

Query fields:

- In the IQL  Ref https://github.com/intermine/intermine/blob/dev/bio/webapp/src/org/intermine/bio/web/logic/GenomicRegionSearchUtil.java#L323

- In ResultRow  Ref https://github.com/intermine/intermine/blob/dev/bio/webapp/src/org/intermine/bio/web/logic/GenomicRegionSearchQueryRunner.java#L212

- In Results table  Ref https://github.com/intermine/intermine/blob/dev/bio/webapp/src/org/intermine/bio/web/logic/GenomicRegionSearchService.java#L1112

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. The whole part will be replaced by InterMine results table.

Ref https://github.com/intermine/intermine/blob/dev/bio/webapp/src/org/intermine/bio/web/logic/GenomicRegionSearchQueryRunner.java#L223

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 activates 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 $MINE/webapp/resources/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

```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 $MINE/webapp/resources/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.

**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/intermine/blob/dev/bio/tutorial/malariamine.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.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Example</th>
<th>Determines</th>
</tr>
</thead>
<tbody>
<tr>
<td>db.production.datasource.serverName</td>
<td>server_name</td>
<td>server name</td>
</tr>
<tr>
<td>db.production.datasource.databaseName</td>
<td>malariamine</td>
<td>database name</td>
</tr>
<tr>
<td>db.production.datasource.user</td>
<td>postgres_user</td>
<td>database username</td>
</tr>
<tr>
<td>db.production.datasource.password</td>
<td>SECRET</td>
<td>database password</td>
</tr>
</tbody>
</table>

The following properties determine the settings for the items database. This database is used during builds only.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Example</th>
<th>Determines</th>
</tr>
</thead>
<tbody>
<tr>
<td>db.common-tgt-items.datasource.serverName</td>
<td>server_name</td>
<td>server name</td>
</tr>
<tr>
<td>db.common-tgt-items.datasource.databaseName</td>
<td>malariamine</td>
<td>database name</td>
</tr>
<tr>
<td>db.common-tgt-items.datasource.user</td>
<td>postgres_user</td>
<td>database username</td>
</tr>
<tr>
<td>db.common-tgt-items.datasource.password</td>
<td>SECRET</td>
<td>database password</td>
</tr>
</tbody>
</table>

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.
### InterMine Documentation, Release

<table>
<thead>
<tr>
<th>Property name</th>
<th>Example</th>
<th>Determines</th>
</tr>
</thead>
<tbody>
<tr>
<td>db.userprofile-production.datasource.serverName</td>
<td>server_name</td>
<td>server name</td>
</tr>
<tr>
<td>db.userprofile-production.datasource.databaseName</td>
<td>malariamine</td>
<td>database name</td>
</tr>
<tr>
<td>db.userprofile-production.datasource.user</td>
<td>postgres_user</td>
<td>database username</td>
</tr>
<tr>
<td>db.userprofile-production.datasource.password</td>
<td>SECRET</td>
<td>database password</td>
</tr>
</tbody>
</table>

#### Web application name and location

<table>
<thead>
<tr>
<th>Property name</th>
<th>Example</th>
<th>Determines</th>
</tr>
</thead>
<tbody>
<tr>
<td>os.production.verboseQueryLog</td>
<td>true</td>
<td>if true, all queries are logged. Defaults to false</td>
</tr>
<tr>
<td>webapp.deploy.url</td>
<td><a href="http://localhost:8080">http://localhost:8080</a></td>
<td>location of tomcat server</td>
</tr>
<tr>
<td>webapp.path</td>
<td>TOMCAT_USER</td>
<td>location of path of webapp</td>
</tr>
<tr>
<td>webapp.manager</td>
<td>TOMCAT_PWD</td>
<td>tomcat username, needed to deploy webapp</td>
</tr>
<tr>
<td>webapp.password</td>
<td><a href="http://www.flymine.org">http://www.flymine.org</a></td>
<td>tomcat password, needed to deploy webapp</td>
</tr>
<tr>
<td>superuser.account</td>
<td>test_user@mail_account</td>
<td>home link; used by client side</td>
</tr>
<tr>
<td>superuser.initialPassword</td>
<td>secret</td>
<td>JavaScript AJAX requests</td>
</tr>
<tr>
<td>project.standalone</td>
<td>true</td>
<td>account name for superuser</td>
</tr>
<tr>
<td>project.title</td>
<td>MalariaMine</td>
<td>password used when account is created</td>
</tr>
<tr>
<td>project.subTitle</td>
<td>An example of InterMine.bio with data from &lt;i&gt;Plasmodium falciparum&lt;/i&gt;</td>
<td>run with associated web site. Defaults to false</td>
</tr>
<tr>
<td>project.releaseVersion</td>
<td>tutorial</td>
<td>name of mine</td>
</tr>
<tr>
<td>project.sitePrefix</td>
<td><a href="http://www.flymine.org">http://www.flymine.org</a></td>
<td>text that appears in the header at the top of the page</td>
</tr>
<tr>
<td>project.helpLocation</td>
<td><a href="http://www.flymine.org/help">http://www.flymine.org/help</a></td>
<td>text that appears at the top of the page next to the mine name</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Example</th>
<th>Determines</th>
</tr>
</thead>
<tbody>
<tr>
<td>mail.host</td>
<td>localhost</td>
<td>mail host to use</td>
</tr>
<tr>
<td>mail.from</td>
<td>account@my_mail_host</td>
<td>“from” email address</td>
</tr>
<tr>
<td>mail.subject</td>
<td>Welcome to MalariaMine</td>
<td>“subject” for email send when account created</td>
</tr>
<tr>
<td>mail.text</td>
<td>You have successfully created an account on MalariaMine</td>
<td>“body” for email send when account created</td>
</tr>
<tr>
<td>feed-back.destination</td>
<td>test_user@mail_address</td>
<td>recipient of feedback form located on bottom of every page</td>
</tr>
</tbody>
</table>

This is the normal mailer. There is a different configuration for SMTP.

1.6. Guide to Customising your Web Application 177
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:
   • `malariamine.properties.dev` - points to the development database and a webapp
2. use the `-Drelease` ant target

   # dev is the suffix on the properties filename
   
   # malariamine/integrate
   ant build-db -Drelease=dev
   
   # malariamine/webapp
   ant default remove-webapp release-webapp -Drelease=dev

Default InterMine properties

In addition to the user properties above, each mine needs two global properties file. These should be in the root directory of the mine - e.g. in the malariamine directory. You should not edit these files.

- `default.intermine.integrate.properties` used by the loading/integration stage
- `default.intermine.webapp.properties` used by the intermine webapp code when running inside Tomcat.

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.
- `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 (e.g. `<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

**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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>branding.images.logo</td>
<td>This image should be 45px by 45px</td>
</tr>
<tr>
<td>branding.colors.header.main</td>
<td>Main colour for your mine, defaults to grey, #595455</td>
</tr>
<tr>
<td>branding.colors.header.text</td>
<td>Text colour for your mine, defaults to white, #fff</td>
</tr>
</tbody>
</table>

**Home page**

Search box (first box on the left)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin.searchBox.title</td>
<td>title of box on left</td>
</tr>
<tr>
<td>begin.searchBox.description</td>
<td>text in the box on the left</td>
</tr>
<tr>
<td>begin.searchBox.example</td>
<td>text in the form field</td>
</tr>
</tbody>
</table>

List upload box (middle box)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin.listBox.title</td>
<td>Title of box</td>
</tr>
<tr>
<td>begin.listBox.description</td>
<td>Text in box</td>
</tr>
<tr>
<td>bag.example.identifiers</td>
<td>Text in form field</td>
</tr>
</tbody>
</table>

Third box

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin.thirdBox.title</td>
<td>Title of box if user is new</td>
</tr>
<tr>
<td>begin.thirdBox.visitedTitle</td>
<td>Title of box if user has visited before</td>
</tr>
<tr>
<td>begin.thirdBox.description</td>
<td>Text in box</td>
</tr>
<tr>
<td>begin.thirdBox.linkTitle</td>
<td>Text for large button</td>
</tr>
<tr>
<td>begin.thirdBox.link</td>
<td>URL for large button</td>
</tr>
</tbody>
</table>

**Tabs**

Templates tagged with each category will appear under the appropriate tab.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin.tabs.1.id</td>
<td>Name of category, eg. Genes</td>
</tr>
<tr>
<td>begin.tabs.1.description</td>
<td>Text for that tab</td>
</tr>
</tbody>
</table>

**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.

If one wants to provide different example identifiers per different types, like Genes, Proteins, SNPs, you can do so by using the following key:
Then, when the user selects a Genes from a dropdown select box for type and choose the Example list link, the text box will be populated keywords associated only with `bag.example.identifiers.gene` key in the config file.

**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.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.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

```mermaid
equation
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 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,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).
```

```bash
~/.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 configuration. 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:

```bash
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 JRTs

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:

1.6. Guide to Customising your Web Application
```java
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

```python
# 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:

<table>
<thead>
<tr>
<th>jwt.verification.strategy</th>
<th>NAMED_ALIAS (default), ANY, or WHITELIST - optional</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>security.keystore.alias.$iss</th>
<th>The alias for the key certificate used to sign the JWT.</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>jwt.alias.whitelist</th>
<th>The comma separated list of aliases to use.</th>
</tr>
</thead>
</table>

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 overridden 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):
The name of the claim that provides the email of the subject. Defaults to `http://wso2.org/claims/emailaddress`

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jwt.publicidentity</code></td>
<td>Used as the <code>iss</code> claim on any tokens the application issues itself. Also, if the tokens received include an <code>aud</code> claim (see <code>aud</code> definition) then this value must match that value for verification to complete. Defaults to your project title.</td>
</tr>
<tr>
<td><code>jwt.verifyaudience</code></td>
<td><code>true</code> or <code>false</code> (default = <code>true</code>). Whether to verify the <code>aud</code> claim.</td>
</tr>
<tr>
<td><code>security.privatekey.password</code></td>
<td>Used to gain access to the private key used by the application for signing its own tokens.</td>
</tr>
<tr>
<td><code>security.privatekey.alias</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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 your webapp directory (`$MINE/webapp`) and then call the following ant task:

```bash
ant verify-jwt
-Drelease=$RELEASE \ # Needed to read the correct properties file
-Dkeystore=$KEYSTORE_LOCATION \ # Needed to read the correct properties file
-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:

```bash
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:

```bash
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:

```properties
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 committed 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.

<table>
<thead>
<tr>
<th>attribute name</th>
<th>purpose</th>
<th>required?</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldExpr</td>
<td>field name</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>human readable name</td>
<td>no</td>
<td>generated automagically</td>
</tr>
<tr>
<td>showInInlineCollection</td>
<td>show field in inline collection (on report pages)</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>showInSummary</td>
<td>add field to query when user clicks on ‘Summary’ button in QueryBuilder</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>showInResults</td>
<td>show field in results table</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>outerInSummary</td>
<td>configure outer-joins when user clicks on ‘Summary’ in QueryBuilder</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>doNotTruncate</td>
<td>don’t truncate display</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>fieldExporter</td>
<td>specify class to export file field</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>sectionOnRight</td>
<td>show on the right side of the page</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>sectionTitle</td>
<td>if sectionOnRight=&quot;true&quot;, title for section on right</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>openByDefault</td>
<td>if sectionOnRight=&quot;true&quot;, whether or not this section should be open</td>
<td>no</td>
<td>false</td>
</tr>
</tbody>
</table>

For example:

```xml
<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>
        <displayer src="attributeLinkDisplayWithField.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.

```xml
<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:

```xml
<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.

```xml
<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"
/>
```

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.

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 `{YOUR_APP}/resources/webapp` 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-precviews 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:

```javascript
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:

```plaintext
head.js.all.MY_JS_OPTIONS = my-mine-options.js
```

**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**
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 (e.g., genes and repeat regions).

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.

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
```

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.
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.

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&value1=2L&constraint2=Gene.organism.name&op2=eq&value2=Drosophila+melanogaster&method=results

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%20Genes&constraint1=Pathway.name&op1=eq&value1=Pentose%20phosphate%20pathway&constraint2=organism.name&op2=eq&value2=Drosophila%20melanogaster&constraint3=Pathway.dataSets.name&op3=eq&value3=KEGG%20pathways%20data%20set&method=list&path=Pathway.genes

**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.


**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 `extraValue` parameter with the organism name, e.g.:


**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.


**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/
Third party tools

Cytoscape network viewer

This tool takes gene interaction data from InterMine and visualises it using [cytoscape.js](http://js.cytoscape.org/), 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:

   ```xml
   <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](https://github.com/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](https://github.com/yochannah/cytoscape-intermine) repo, and the [standalone app demo page](http://yochannah.github.io/cytoscape-intermine/)

**Dependencies:** This tool uses [imjs](https://github.com/intermine/imjs) to query the data, and [imtables](https://github.com/intermine/im-tables) 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:

```xml
<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:

```xml
<action path="/initEsynListDisplayer" type="org.intermine.bio.web.EsynListDisplayer"/>
```

2. add the following entries to your `'tiles-def-model.xml` file:

```xml
<definition name="esynListDisplayer.tile" path="/model/esynListDisplayer.jsp" controllerUrl="/initEsynListDisplayer"/>
```

Galaxy

Enable Galaxy export

Add to `'web.properties` file, replacing FlyMine with your Mine name:

```properties
# in web.properties
galaxy.display = true
galaxy.disabledMessage = Galaxy export is disabled.
galaxy.baseurl.default = http://main.g2.bx.psu.edu
galaxy.url.value = /tool_runner?tool_id=flymine
galaxy.welcomeMessage = <b>Welcome to FlyMine, GALAXY users</b><br/>You can run queries by clicking on the 'Templates' tab at the top of this page. You can end the query results to Galaxy from the 'EXPORT' menu in the results page.
```

Update Struts config

```xml
# MINE/webapp/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"/>
```

```xml
# MINE/webapp/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"/>
```
# MINE/webapp/resources/struts-config-model-form.xml

```xml
<form-bean name="galaxyExportForm" type="org.intermine.bio.web.struts.GalaxyExportForm"/>
```

## Customization

Properties located in the `'web.properties'` file.

<table>
<thead>
<tr>
<th>parameter</th>
<th>purpose</th>
<th>required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>enable Galaxy export</td>
<td>yes[1]</td>
</tr>
<tr>
<td>disabledMessage</td>
<td>displayed when Galaxy export is disabled</td>
<td>yes</td>
</tr>
<tr>
<td>baseurl.default</td>
<td>base url of Galaxy server</td>
<td>yes[2]</td>
</tr>
<tr>
<td>url.value</td>
<td>tool runner url</td>
<td>yes[3]</td>
</tr>
<tr>
<td>welcomeMessage</td>
<td>displays on the homepage when coming from Galaxy</td>
<td>yes</td>
</tr>
</tbody>
</table>

```java
# galaxy
## set to "false" to disable galaxy
galaxy.display = true
galaxy.disabledMessage = Galaxy export is disabled.
galaxy.baseurl.default = http://main.g2.bx.psu.edu
galaxy.url.value = /tool_runner?tool_id=flymine
galaxy.welcomeMessage = <b>Welcome to FlyMine, GALAXY users</b><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.
```

### 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.

Expression data type is an extension of InterMine core model. It is addressed in `modmine/dbmodel/resources/modencode-metadata_additions.xml`

```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 selects a proper webpage 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:

```xml
<action path="/initHeatMap" type="org.modmine.web.HeatMapController" />
<action path="/heatMap" forward="heatMap.page" />
```

```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. However canvasXpress is particular designed to display genomics data, D3 is for a broader use.
InterMine Documentation, Release

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-1.11.5.

NOTE: If you already have a JBrowse installation working and just want to embed it in your report pages then see: Embedding JBrowse.

Build Your InterMine Database

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.

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:

```json
{
    "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

```java
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.

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

```java
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.

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:

```java
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:

```java
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....

Furthermore, 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.

**Import/Export XML**

Import and export XML functionalities allow you to:

- export all tracks from the userprofile database into an XML file
- import all tracks from an XML file into the userprofile database

The import functionality generates the tracker tables, if these haven’t been created before.

When you release a webapp, you may need to recreate the userprofile database without losing all tracks saved until that moment. Before running `create-db-userprofile`, execute `write-userprofile-xml` in order to save all the tracks into the `userprofile.xml` file.

The file has the following structure:

```
<userprofiles>
<tracks>
<templatetracks>
<templatetrack templatename="fourConstraints" username="butano@flymine.org" sessionidentifier="18FB96389B8C44817780B1B778C6F1C2" timestamp="2011-05-06 14:43:42.779"/>
```

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:

```plaintext
Sitemap: sitemap_index.xml

User-agent: *
Disallow: /

User-agent: Googlebot
Disallow: /release-8.2/
Disallow: /release-8.1/

User-agent: Slurp
Disallow: /release-8.2/
Disallow: /release-8.1/

User-agent: msnbot
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:

```plaintext
# 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.
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.

<table>
<thead>
<tr>
<th>tag</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>im:public</td>
<td>make list/template viewable by all users</td>
</tr>
<tr>
<td>im:frontpage</td>
<td>put list on home page</td>
</tr>
<tr>
<td>im:converter</td>
<td>template used in generating links in the ‘Convert’ section on the list analysis page</td>
</tr>
<tr>
<td>im:aspect:CategoryName</td>
<td>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</td>
</tr>
<tr>
<td>im:report</td>
<td>allows template to be displayed on report or list analysis page</td>
</tr>
<tr>
<td>im:order:n</td>
<td>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</td>
</tr>
</tbody>
</table>

Fields and collections

The SuperUser can change how fields are displayed by adding tags via the report page.

<table>
<thead>
<tr>
<th>tag</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>im:hidden</td>
<td>hides the field/collection</td>
</tr>
<tr>
<td>im:summary</td>
<td>add collection to ‘Summary’ section of report page</td>
</tr>
<tr>
<td>im:aspect:CategoryName</td>
<td>collection appears underneath category</td>
</tr>
</tbody>
</table>

Classes

The SuperUser can change how classes are displayed by adding tags via the model browser.

<table>
<thead>
<tr>
<th>tag</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>im:aspect:CategoryName</td>
<td>class appears on aspect page</td>
</tr>
<tr>
<td>im:preferredBagType</td>
<td>class appears first in the class selection</td>
</tr>
</tbody>
</table>

15Editable by all admins
**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 convert 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

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" longDescription="" comment="" >
  <query name="Gene_To_Protein_Type_Converter" model="genomic" view="Gene.id Gene.proteins.id" longDescription="" sortOrder="Gene.id asc" >
    <node path="Gene" type="Gene"></node>
    <node path="Gene.id" type="Integer">
      <constraint op="=" value="0" description="Gene.id" identifier="Gene.id" editable="true" code="A" ></constraint>
    </node>
  </query>
</template>
```

**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. Use this command, replacing HOST and USER with the correct values:

```
$ createdb userprofile -h HOST -u USER
```

These commands are needed in the webapp to initialise a userprofile database:

```
# in <MINE_NAME>/webapp
$ ant build-db-userprofile
```

**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

```sql
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**

1. cron job on production runs this script:
   - `userprofile_backup_cron`
   - dumps db, only saves copy if different from previous version
   - updates sym links
2. `userprofile_backup_cron` script runs this script:
   - `userprofile_backup`
   - does the dumping

**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:

  ```properties
  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.
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/)

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

If you are using Tomcat 7, please make sure you have configured it correctly. See Tomcat

Force recompile

Run this command in your `webapp` directory:

```
$ ant clean-all
```

Verify `/build` is gone from your `webapp` directory.
Re-release webapp

$ ant default remove-webapp release-webapp

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:

$ cd flymine/webapp
$ ant 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 git/intermine/flymine/webapp/build/javadoc.

Generating Intermine-wide Javadoc

If you just want to browse the docs, you can see the most recent version at http://intermine.org/intermine/. If you’d like to generate it yourself, read on.

Dependencies

The Javadoc depends on the database model under bio/testall being built first. To do this, from the intermine root directory:

$ cd bio/test-all/dbmodel
$ ant clean-all
$ ant build-db

Building

Once you’ve successfully run build-db above, you’ll need to change to the imbuild/javadoc directory and generate the documentation:

$ cd ../../imbuild/javadoc
$ ant clean
$ ant

That's it! You’ll be able to see the documentation under intermine/imbuild/javadoc/build/javadoc.
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.

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 your_mine/webapp/resource/web.properties
```
InterMine Documentation, Release

services.defaults.query = <query model="genomic" view="Gene.secondaryIdentifier Gene.symbol Gene.primaryIdentifier ... ASC" ><constraint path="Gene.organism.name" op="=" value="Drosophila melanogaster" code="A" /></query>

Overwrite any JSP

When the webapp is compiled, the order of projects is:

1. intermine/webapp
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.

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.

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 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).

How Do I?

For the kinds of things you will probably be wanting to do, see howtows
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](http://iodocs.apps.intermine.org)

- **Java**  Download | Docs | Source
- **Perl Client**  Download | Docs | Source
- **Python Client**  Download | API Examples | Source
- **Ruby Client**  Download | Docs | Source
- **JavaScript Client (for Browser and node.js)**  Download | Docs | Source
- **R Client**  Download | Docs | Source

Authentication

Authenticated web services are accessed via tokens: either 24-hour anonymous tokens or permanent user API key tokens. See `/web-services/authentication`

Embedding InterMine components

The following are libraries that compile to JavaScript which can be embedded on *any* webpage.

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.
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.

```html
<!-- 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.

```javascript
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.

```javascript
// 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.

```javascript
// 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.

```javascript
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.

```javascript
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 occasions 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:

```javascript
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:

```javascript
var options = {
    resultsCb: function(pq) {
        // ...
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Clicking on Create list button in a popover window:

```javascript
var options = {
    listCb: function(pq) {
        // ...
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

I want to hide the title or description of a widget.

```javascript
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`:

```javascript
// 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.
```
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:

\[
P = \frac{(M \text{ choose } k)(N-M \text{ choose } n-k)}{N \text{ choose } n}
\]

- **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

**Note:** The relevant Java source.

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:

\[
CCGO = \frac{LGO}{LWH} \frac{NGO}{NWG}
\]

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. 
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

1.8. Embedding InterMine components  213
Boyle EI, Weng S, Gollub J, Jin H, Botstein D, Cherry JM, Sherlock G.  
PubMed id: 15297299

Controlling the false discovery rate: a practical and powerful approach to multiple testing  
Benjamini, Yoav; Hochberg, Yosef  

Augmentation Procedures for Control of the Generalized Family-Wise Error Rate and Tail Probabilities for the Proportion of False Positives  
vander 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.  

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.

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique id used by javascript only. Spaces not allowed.</td>
<td>unique_id</td>
</tr>
<tr>
<td>pathStrings</td>
<td>which collection to use in the widget</td>
<td>Gene.homologues[type=orthologue].homologue.organism</td>
</tr>
<tr>
<td>exportField</td>
<td>which field from the objects in your list to export</td>
<td>primaryIdentifier</td>
</tr>
<tr>
<td>typeClass</td>
<td>types of lists that should display this widget. Use the simple class name</td>
<td>Gene</td>
</tr>
</tbody>
</table>

The following are optional attributes:

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>appears at the top of the widget</td>
<td>Orthologues</td>
</tr>
<tr>
<td>description</td>
<td>description of the widget</td>
<td>Counts of orthologues</td>
</tr>
<tr>
<td>displayField</td>
<td>which fields from the objects in the collection (in the above example,</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>Gene.proteins) to display, eg. primaryAccession</td>
<td></td>
</tr>
<tr>
<td>columnTitle</td>
<td>heading for the “count” column</td>
<td>Orthologues</td>
</tr>
<tr>
<td>externallink</td>
<td>link displayed next to first column, identifier will be appended to link</td>
<td></td>
</tr>
<tr>
<td>externallink</td>
<td></td>
<td></td>
</tr>
<tr>
<td>views</td>
<td>path fields display in the query running when the user clicks on the widget</td>
<td>symbol</td>
</tr>
</tbody>
</table>

Graph/Chart widgets

Graph widgets display datasets in graphical format.
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

<table>
<thead>
<tr>
<th>Organism.name</th>
<th>Orthologues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homo sapiens</td>
<td>60</td>
</tr>
<tr>
<td>Mus musculus</td>
<td>53</td>
</tr>
<tr>
<td>Rattus norvegicus</td>
<td>53</td>
</tr>
<tr>
<td>Danio rerio</td>
<td>41</td>
</tr>
<tr>
<td>Drosophila pseudoobscura</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila simulans</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila erecta</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila virilis</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila yakuba</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila sechellia</td>
<td>8</td>
</tr>
<tr>
<td>Drosophila persimilis</td>
<td>8</td>
</tr>
</tbody>
</table>
Figure 1.24: An example chart widget of BDGP Expression Patterns in FlyMine.
<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique id used by javascript only. Spaces not allowed.</td>
<td>Column_id, &quot;BarChart&quot;</td>
</tr>
<tr>
<td>graphTypeError</td>
<td>which type of chart to render</td>
<td>Gene</td>
</tr>
<tr>
<td>startClass</td>
<td>it’s the root class for all the paths specified in the configuration.</td>
<td>mRNAExpressionResults.stageRange, mRNAExpressionResults.expressed</td>
</tr>
<tr>
<td>typeClass</td>
<td>type of lists that should display this widget. Use the simple class name.</td>
<td></td>
</tr>
<tr>
<td>categoryPath</td>
<td>Must be attribute. We can specify the subclass using the syntax path[subclass type].</td>
<td></td>
</tr>
<tr>
<td>seriesPath</td>
<td>the series path. This has to be an attribute. We can specify the subclass using the syntax path[subclass type].</td>
<td></td>
</tr>
<tr>
<td>seriesValues</td>
<td>the values of different series. Case sensitive. You can specify boolean values</td>
<td>true, false or Up, Down</td>
</tr>
<tr>
<td>seriesLabels</td>
<td>the labels displayed on the graphs to distinguish inside a category the different series</td>
<td>Expressed, Not Expressed or Up, Down</td>
</tr>
<tr>
<td>views</td>
<td>attributes paths displayed when the user clicks an area on the graph</td>
<td>name, organism.name</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>appears at the top of the widget</td>
<td>BDGP expression patterns</td>
</tr>
<tr>
<td>description</td>
<td>description of the widget</td>
<td>Expression patterns</td>
</tr>
<tr>
<td>domainLabel</td>
<td>Label displayed on x-axis in the ColumnChart (on y-axis in the BarChart)</td>
<td>Stage</td>
</tr>
<tr>
<td>rangeLabel</td>
<td>Label displayed on y-axis in the ColumnChart (on x-axis in the BarChart)</td>
<td>Gene count</td>
</tr>
<tr>
<td>filterLabel</td>
<td>Label for filter form field</td>
<td>Organism</td>
</tr>
<tr>
<td>filters</td>
<td>the values for the filter, set in the dropdown</td>
<td>All, KEGG pathways, Reactome data</td>
</tr>
<tr>
<td>listPath</td>
<td>the path used to build the bag constraint</td>
<td>FlyAtlasResult.material</td>
</tr>
<tr>
<td>constraints</td>
<td>separated by comma, case sensitive, must be attributes, operator can be = or !=</td>
<td>organism.name=[Organism]</td>
</tr>
</tbody>
</table>

**Note:** The graphs use Google Visualization API.

---

16 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.

17 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.

18 Optional if the startClass contains the bag type class.

19 For the values we can use static values or the selected filter value using the syntax: path constraint = [filter identifier].

20 organism’s name matching with the value selected in the filter with filterLabel ‘Organism’
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.

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique id used by JavaScript only. Spaces not allowed.</td>
<td>unique_id</td>
</tr>
<tr>
<td>startClass</td>
<td>Root class for all the paths specified in the configuration. Use simple name (e.g. Gene)</td>
<td>Gene</td>
</tr>
<tr>
<td>startClassDisplay</td>
<td>Field displayed when user clicks on the widget on ‘Matches’ column</td>
<td>primaryIdentifier</td>
</tr>
<tr>
<td>typeClass</td>
<td>Type of lists that should display this widget. Use the simple class name.</td>
<td>goAnnotation.ontologyTerm.parents.name</td>
</tr>
<tr>
<td>enrich</td>
<td>Field to be enriched, displayed in the widget in the first column</td>
<td>21. goAnnotation.ontologyTerm.parents.name</td>
</tr>
<tr>
<td>views</td>
<td>attributes paths displayed when the user clicks on View results button</td>
<td>symbol,organism.name</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>attribute</th>
<th>purpose</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>appears at the top of the widget</td>
<td>Gene Ontology</td>
</tr>
<tr>
<td>description</td>
<td>description of the widget</td>
<td>Enrichment</td>
</tr>
<tr>
<td>label</td>
<td>heading for the column</td>
<td>GO terms enriched.</td>
</tr>
<tr>
<td>exportedLink</td>
<td>field displayed next to first column</td>
<td>GO Term</td>
</tr>
<tr>
<td>filters</td>
<td>extra filters to add to the display</td>
<td>google</td>
</tr>
<tr>
<td>filterLabel</td>
<td>label for filter form field</td>
<td>organism.name=[list]</td>
</tr>
<tr>
<td>enrichIdentifier</td>
<td>identifier for the row displayed, if not specified, enrich field used</td>
<td>goAnnotation.ontologyTerm.parents.name</td>
</tr>
<tr>
<td>constraints</td>
<td>constraints separated by comma. The paths have to be attributes. The operator can be = or != 24.</td>
<td>organism.name=[list]</td>
</tr>
<tr>
<td>constraintsForView</td>
<td>constraints separated by comma used for building the query executed when the user clicks on the widget on ‘Matches’ column</td>
<td>results.expressed = true</td>
</tr>
</tbody>
</table>

**Examples**

See other mines’ config files for more examples, eg:

- branches/intermod_workshop/flymine/webapp/resources/webapp/WEB-INF/webconfig-model.xml in FlyMine
- branches/intermod_workshop/modmine/webapp/resources/webapp/WEB-INF/webconfig-model.xml in modMine
- branches/intermod_workshop/metabolicmine/webapp/resources/webapp/WEB-INF/webconfig-model.xml in metabolicMine

---

21 You have to specify only one field. Specify the subclass using the syntax path[subclass type].
22 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.
23 Specify only one. This has to be an attribute. Used in the results table. Specify the subclass using the syntax path[subclass type].
24 Case sensitive. For the values we can use: static values the selected filter value using the syntax: path constraint = [filter identifier] only the value contained in the list.
Gene Ontology Enrichment
GO terms enriched for items in this list.

Test Correction: Holm-Bonferroni
Max p-value: 0.05
Ontology: biological_process

<table>
<thead>
<tr>
<th>GO Term</th>
<th>p-Value</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>regulation of transcription, DNA-dependent</td>
<td>8.613623e-22</td>
<td>23</td>
</tr>
<tr>
<td>regulation of RNA biosynthetic process</td>
<td>8.613623e-22</td>
<td>23</td>
</tr>
<tr>
<td>transcription, DNA-dependent</td>
<td>6.709974e-21</td>
<td>23</td>
</tr>
<tr>
<td>RNA biosynthetic process</td>
<td>7.113099e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of RNA metabolic process</td>
<td>7.113099e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of macromolecule biosynthetic process</td>
<td>9.236209e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of cellular macromolecule biosynthetic process</td>
<td>9.236209e-21</td>
<td>23</td>
</tr>
<tr>
<td>regulation of transcription from RNA polymerase II promoter</td>
<td>1.271350e-20</td>
<td>19</td>
</tr>
</tbody>
</table>

Figure 1.25: An example enrichment widget of Gene Ontology in FlyMine.
Background population

In the enrichment 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

Widget service  Create a new Widgets instance pointing to a service:

```javascript
var widgets = new intermine.widgets("http://beta.flymine.org/query/service/");
```

Choose a widget  Choose which widget(s) you want to load:

```javascript
// 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.

Note: Online example can be visited at tinkerbin.

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/query/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/query/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:

   ```javascript
   // 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
   <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/query/service/');
       // Load all Widgets:
       Widgets.all('Gene', 'myList', '#all-widgets');
     });
     </script>
   </head>
   ```
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.

**Apps/A**

New in version 1.2.3.

This document describes how to develop and host your own Apps.

**Middleware**

Apps/A Middleware is a glue that makes serving Apps possible. It depends on Node.js so make sure it is installed. Afterwards either checkout or download & unzip the repo then install its dependencies and run the example:

```
$ wget https://github.com/intermine/apps-a-middleware/archive/master.zip
$ unzip master.zip
$ cd apps-a-middleware-master/
$ npm install
$ PORT=1234 node example/index.js
```

Visiting 127.0.0.1:1234 should now show a bunch of Apps on a page. They are coming from InterMine Apps/A Sources. The next section will describe how to develop a custom app like one of those on the page.

**Developing an App**

The middleware accepts one or more paths to a directory with Apps. Above, we have seen an example of loading such a directory. Let us have a look at one of the Apps, a publications-displayer.

The middleware knows of and can load this App because we have created a folder that has a name that can be used in a URL. If we were to add spaces, for example, that would be a big no no.

Next we have a configuration stashed away in a config.js file. This file exports config in a JSON-like syntax. It will have a header with some properties like author, tile, description etc. The more interesting point is the dependencies section. InterMine JavaScript API Loader describes how this section works. It is here that you define what kind of CSS and JS dependencies your App has. These will be automatically resolved before loading the App on a page.

You can fill the file’s config section to show whoever embeds your App what sort of configuration they should be providing. The idea is as follows:

1. The App’s config should cover dependencies and show example config with comments.
2. The middleware should provide mine specific settings, like pathQueries to run.
3. The client instantiating the App should provide “page”-specific settings like which specific Gene we want to show visualization for etc.

Next up is the body of the App in a presenter.[js|ts|ls|coffee] file. The extensions allure to the fact that you can use the following languages to write your App in:

### 1.8. Embedding InterMine components
1. plain vanilla JavaScript
2. TypeScript; including having type definitions in a separate file like `lib.d.ts` and targeting ECMAScript 5
3. LiveScript
4. CoffeeScript

This file will have all the logic needed to “do something”. It needs to fulfill the following interface:

```javascript
exports.App = (function() {
    function App(config, templates) {}
    App.prototype.render = function(target) {};
    return App;
})();
```

Or the same in TypeScript:

```typescript
export class App {
    constructor(config: Object, templates: Object) {
    }
    render(target: string): void {
    }
}
```

Or the same in CoffeeScript:

```coffeescript
class exports.App
    constructor: (config, templates) ->
    render: (target) ->
```

Or the same in LiveScript:

```livescript
class exports.App
    (config, templates) ->
    render: (target) ->
```

The constructor takes two parameters:

- `config` This is an amalgamated config that the user and the middleware have provided.
- `templates` This will be an Object containing a map from template name to an object. More on them later.

The render function takes just one parameter:

- `target` A string that tell you where to render/display your content to.

Next up are templates. They are the place where you put your HTML that will be rendered. You have a choice between two templating languages:
1. **eco** which allows you to use CoffeScript logic inside the template. To use this language save the file as `.eco`. An eco template wraps all the logic needed to execute it, as an example:

```javascript
class exports.App

constructor: (config, @templates) ->

render: (target) ->
  $(target).html @templates[template_name_with_extension] { 'some': 'data', 'right': [ 'here' ] }
```

2. **Hogan.js** is an implementation of the language **mustache**. To use this variant, save your file as `.hogan`. You still need to include a reference to the Hogan library in your App and then initialize and use them as follows:

```javascript
class exports.App

constructor: (config, @templates) ->

render: (target) ->
  $(target).html (new Hogan.Template(@templates['template.hogan'])).render { 'some': 'data', 'right': [ 'here' ] }
```

Finally we might want to style our app. Usually a main style will be defined by a CSS framework required in the config file, but there is always place for that special something. To define a custom style **guaranteed** to be applicable to your App only, save a CSS or Stylus file as `style.[css|styl]`.

To run it all refer to the `example/index.js` and `example/public/index.html` files in the middleware repo.

**Q & A**

**How do I change which file will export my root App?**

Extend your `config.js` with the following key, value pair:

```javascript
module.exports = {
  "appRoot": "app/index" // points to say index.js in "app" folder
}
```

**How can I use modules across folders?**

Use the **CommonJS Modules/1.1** require pattern, for example as follows (in TypeScript):

```javascript
import models = module("models");

var pete = new models.Person('pete');
```

And in `models.ts`:

```javascript
export class Person {
  constructor(name: string) {
  }
}
```

**See also:**

Take a look at the **choose-list** app in the **intermine-apps-a** repo. It shows an example of how different modules can be required.

---

1.8. Embedding InterMine components
I am using require.js and the app just blew up

Each app gets its own internal implementation of require to load its and only its modules. If you have something on `window.require` it will not get overwritten but it will also not be used by us. Your apps see our version of require, not any other higher up. This is so that multiple apps can live peacefully on a page without deviating from the require spec and having our own prefixes and app silos.

In the future, we will also want to get the compiled app once into the browser and then require it multiple times on a page if need be. Using vanilla module loader would then not work as we would not be getting new instances of modules per app instance.

Suggestions are welcome.

A source file I am using is not compiling

Check the head of the following file to see the regex rules that the builder uses when going through your app sources. For example:

```javascript
rules = [
    [ /^((presenter|app)(\.(coffee))|(\/.*)\.(coffee))/g, 'module', 'coffeescript' ]
    [ /^((presenter|app)(\.(js))|(\/.*)\.(js))/g, 'module', 'generic' ]
    [ /^((presenter|app)(\.(ls))|(\/.*)\.(ls))/g, 'module', 'livescript' ]
    [ /^(.*)\.(css)/g, 'style', 'generic' ]
    [ /^(.*)\.(styl)/g, 'style', 'stylus' ]
    [ /^(.*)\.(eco)/g, 'template', 'eco' ]
    [ /^(.*)\.(hogan)/g, 'template', 'hogan' ]
]
```

Would like to add another rule/filetype? Define its rule and add a new handler in `apps-a-middleware/builder/types`.

Why do you keep talking about a middleware

Well, the service you are using extends any existing service you are offering to your users. It would not make much sense to fire up a new service instance per “a thing” that you want to offer to your users. By using a middleware concept, you create one Node.js service on one port and it offers various functionalities as specified by its middlewares.

Why the name Apps/A?

You can just call it “apps”, but as a developer you need some form of a namespace to go on, so we use the Latin alphabet for suffixes. In the future/different version will be called Apps/B and so on.

Which language should I use for the app?

Down to your preference:

**JavaScript** Oldie but goldie. There is no compilation step involved so what you see is what you get.

**CoffeeScript** Sports a pythonesque syntax and makes common operations for you (far) easier. Your code is less verbose. The source you see gets compiled to JavaScript so if you run into problems you are still debugging JS.
TypeScript  You get the added benefit of types if you want to use them. This is beneficial especially if you have an editor with “IntelliSense” like WebStorm. Ultimately this is where a lot of JavaScript is headed with classes and modules. You write more code but you can also define interfaces for what you expect to be provided to you. If you work with multiple different people this might be beneficial as they might be alerted when they break something.

Why is the config in JavaScript if my App can be in *Script?

This is so they can see what the example config looks like. Someone embedding your App does not need your implementation language, but they can/should know vanilla JavaScript.

Apps/B

New in version 1.2.3.

This document describes how to build JavaScript components using the Apps/B Component.io builder.

Quick Start

Globally install the npm package:

```sh
$ sudo npm install apps-b-builder -g
```

Then specify the input and output path to build a component:

```sh
$ apps-b ./src/ ./build/
```

Create a component

A component consists of one component.json config file and one or more source file. Script source files use the CommonJS Modules/1.1 implementation so you use `require` and `module.exports` to link between modules & components. This is a standard in the Node.js community.

Component config file

To write a component config file in JSON refer to the standard.

```json
{
   "name": "app",
   // Which file do we require as the main file.
   "main": "app.js",
   "version": "1.0.0",
   // Other components.
   "dependencies": {
      "visionmedia/superagent": "*",
      "necolas/normalize.css": "*",
      "component/marked": "*"
   },
   "scripts": [
      "app.coffee",
      "template.eco"
}
Supported types

Have a look into the `test/fixtures` directory for examples of supported filetypes:

1. CoffeeScript; compile-to-JS language, goes into the `scripts` section
2. CSS, goes into the `styles` section
3. Eco; a templating language, goes into the `scripts` section
4. JavaScript, goes into the `scripts` section
5. Literate CoffeeScript; mix Markdown and CS syntax, goes into the `scripts` section
6. Stylus; a CSS preprocessor including nib CSS3 extensions, goes into the `styles` section

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:

```javascript
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:

```javascript
// 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.

---

**1.8. Embedding InterMine components**
This directory will be automatically created and will contain libraries we have requested through the Bower system.

Contains an example of our app in use.

Source files that our code will consist of.

Will contain a listing of libraries we want to download using Bower.

Lists libraries we will need to compile and build our app with.

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:

```json
{
   "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
```

Now we want to fetch libraries that our app, when running, will depend on.

Edit the bower.json file like so:

```json
{
   "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:
```

1.8. Embedding InterMine components
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">
```
This file does not do anything else other than 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'
```
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 requiring 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:

```html
<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`:

```html
<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>
```
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:

```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:

```html
<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:

```coffeescript
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:

```coffeescript
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`:

```coffeescript
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
    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.

---

**1.8. Embedding InterMine components**
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 initializing 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 embryonic epsilon- and fetal G gamma-globin genes in transgenic mice.",
    "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">
```

1.8. Embedding InterMine components 239
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 represent 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.
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 successful, 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:

```javascript
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:

```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>'
        }
      });
    });
```
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!):

```html
<!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.
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:

---

1.8. Embedding InterMine components 243
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:

```bash
$ 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:

```json
{
  "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:

```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:
```

1.8. Embedding InterMine components
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:

```html
<!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>
    <div id="app"></div>
    <script>
        // Once scripts have loaded.
        $(function() {
            // ...show the app.
            require('em')({
                'el': '#app',
                'service': 'http://newvegas:9200',
                'index': 'publications',
                'type': 'publication',
                'query': 'breast size exercise cancer'
            });
        });
    </script>
</body>
```

1.8. Embedding InterMine components
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:

```coffee
module.exports = (opts) ->
  # Explode ejs options.
  { service, index, type } = opts

  # Init the ejs client.
  ejs.attr { index, type, 'client': new $.es.Client({ 'hosts': service }) }

  # Start routing.
  new Routing opts.el
  do can.route.ready

  # Have we launched on the index?
  if can.route.current('')
    # Manually change the query to init the search.
    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:

```coffee
# Router switching between pages.
Routing = can.Control

# Load the components.
init: ->
  ( 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')
    @render template, doc, "#{title} - ElasticMed"

# Find the document.
doc = null
# Is it in results?
if (docs = results.attr('docs')).length
  docs.each (obj) ->
    # Found already?
    return if doc
    # Match on oid.
    doc = obj if obj.attr('oid') is oid
  return fin(doc) if doc
```

1.8. Embedding InterMine components 249
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 explicitly 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> to autocomplete or <kbd>Enter</kbd> to search.</p>

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:

Now for the template that gets rendered on a detail page, in src/templates/page/detail.mustache:
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.current }}">
    {{ #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 }}
    </ul>
  </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

Apps/* Comparison

We have developed a variety of builders that can be used to make the development and deployment of your JavaScript components easier.

Follow the guide below to decide which approach is the best for your situation. If in doubt use Apps/C.

Apps/A

url  https://github.com/intermine/apps-a-middleware
examples  https://github.com/intermine/intermine-apps-a
languages  JavaScript, CSS, Stylus, CoffeeScript, LiveScript, TypeScript, Eco, Hogan
what  Middleware for loading apps, resolving vendor dependencies at runtime and sandboxing CSS styles.
when  You are loading a lot of apps on a page and they are likely going to depend on the same vendor library dependencies. These are resolved at runtime, in the browser, and thus you can save on requests by sharing the same libraries over and over. You want to automatically prefix your CSS styles so that your app does not clash with the rest of the apps/styles on the page. You want to provide an example config that people are to use with your app, but want to give developers the ability to override. You want to create a repository of apps and have a delivery mechanism ready (git is supported).
issues  Dynamic resolving of libraries at runtime is nice but it takes one app to crash them all. You could also encounter a situation where one app needs say jQuery 1.9 and another jQuery 2.0, try solving that one.
alternatives  Use Apps/C together with pommejs to load apps in an iframe so it has no chance of colliding with the other resources on the page.

Apps/B

url  https://github.com/intermine/apps-b-builder
examples  NA
languages  JavaScript, CSS
what  Builder for ComponentIo based packages.
when  Use it when you want as small a build as possible relying on packages in the ComponentIo ecosystem. As all vendor dependencies take the CommonJS approach to loading files, you will unlikely going to encounter difficulty running your app besides other apps on the page.
issues  One has to develop in JavaScript, CSS and cannot use languages such as CoffeeScript if your app is to be available through the ComponentIo ecosystem.
alternatives  Use Apps/C and use the grunt-component-io task to make use of libraries available through the ComponentIo ecosystem.

Apps/C

url  https://github.com/intermine/grunt-apps-c
examples  https://github.com/intermine/intermine-apps-c
languages  JavaScript, CoffeeScript, Stylus, Eco, Mustache, JSON and more


when  You want to use the de-facto standard in building & bundling JavaScript applications; this gives you access to big ecosystems of libraries to choose from.

issues  You need to prefix your CSS styles yourself (use grunt-rework) and your bundled packages can clash if you are depending on libraries that export themselves globally (on window object). To alleviate these issues you can load your app using the pommejs library. Since none of your builds share resources (unless you make a mega-build), your apps can get bulky.

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 be in brackets.

The initial Page Size

This can be configured on a table by table basis when the table is initialised:

```javascript
$('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 /webapp/properties/javascript-properties.

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 unnecessary 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:

```javascript
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:

```javascript
(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 query for the object.

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:


### 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', {
});

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.

**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.

```html
<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:

```javascript
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. Embedding InterMine components
InterMine JavaScript Library

Please refer to this repo for more information.

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.

InterMine API Description

This section describes the public API definitions of parts of the InterMine system.

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 satisfies 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:

```json
{
  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 specify 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:

```xml
<constraint path="Gene" op="LOOKUP" value="eve"/>
```

```xml
{  
  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:

```xml
<constraint path="Gene" op="LOOKUP" value="eve" extraValue="D. melanogaster"/>
```

```xml
{  
  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:

```xml
<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"/>
```

```xml
{  
  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 !=.

```xml
<constraint path="Gene.homologues.homologue" op="=" value="Gene.pathways.genes"/>
<!-- or -->
<constraint path="Gene.homologues.homologue" op="!=" value="Gene.pathways.genes"/>
```

```xml
{  
  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.

```xml
<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:

```xml
<query model="genomic" view="Gene.symbol" sortOrder="Gene.length DESC Gene.name ASC"/>
{
  select: ["Gene.symbol"],
  sortOrder: [["length", "DESC"], ["name", "ASC"]]
}
```

Support

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

Troubleshooting tips

This page describes what to do if you encounter problems when installing or maintaining InterMine. Please feel free to :docs:`/about/contact-us` with any questions you may have.
Error messages

If you encounter a problem when running Ant, try adding the verbose flag:

```bash
$ ant -verbose
```

This should output a more useful error message.

Logs

Data warehouse

When integrating data, usually the errors are in intermine.log file in the directory you are in, eg. `/integrate` or `/dbmodel`

Webapp

In order for the intermine webapp to write logs, you must correctly set the `webapp.logdir` property in your mine properties file. The property must be an absolute path to a directory on the tomcat machine, writable by tomcat.

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:

```bash
$ 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:

```bash
$ ant run-iql-query -Dquery='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/PREFIX/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 49.0)

This means that your version of Java is too old, you need at least Java 6 to run !InterMine.

can’t open datasource

java.lang.RuntimeException: can’t open datasource for {platform=PostgreSQL, datasource.dataSourceName=db.flatmodeunittest, ...

Errors of this type mean there was a problem accessing a database, in this example with db.flatmodeunittest. Either the database specified in the xxx.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 - for database: db.bio-fulldata-test

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 postgres.conf file and restart postgres. Try 50 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 ANT_OPTS environment variable. We recommend 1000M as a minimum, more is often needed during dataloading. Your ANT_OPTS variable should include the following:

$ echo $ANT_OPTS
$ -Xmx1000M -XX:MaxPermSize=256M

OutOfMemoryError: PermGen space

java.lang.OutOfMemoryError: PermGen space

This occurs when Java has run out for !PermGen space - where it stores !ClassLoaders and Class definitions, etc. It can be fixed by increasing the -XX:MaxPermSize value in your ANT_OPTS environment variable, the default is 128M, trying 256M would be best. The ANT_OPTS variable should include the following:

$ echo $ANT_OPTS
$-Xmx1000M -XX:MaxPermSize=256M

Can’t find class name ClassName

Exception caught: java.lang.IllegalArgumentException: Unknown class name Protein in package org.intermine.model.bio

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 sourcename_additions.xml files to see if the class is listed, only the additions files for sources lists on project.xml when ant build-db 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
bio/sources/protfeatures/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) "${dec.boolean}"

Add this to your Tomcat startup.sh script:

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

See our section on 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.

Contact us
To contact the InterMine Team:

email info [at] intermine [dot] org
twitter https://twitter.com/intermineorg
blog https://intermineorg.wordpress.com/
Upgrade Instructions

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

```bash
$ git pull upstream
```

If you don’t have a git repo yet, see Get the Software for details.

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

<table>
<thead>
<tr>
<th>class</th>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>gene1</td>
<td>participant1</td>
</tr>
<tr>
<td></td>
<td>gene2</td>
<td>participant2</td>
</tr>
<tr>
<td></td>
<td>relationshipType (Term)</td>
<td>relationshipType (String)</td>
</tr>
<tr>
<td></td>
<td>allInteractors (Gene)</td>
<td>allInteractors (Interactor)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stoichiometry</td>
</tr>
<tr>
<td>InteractionDetail</td>
<td>allInteractors (Gene)</td>
<td>allInteractors (Interactor)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>role</td>
</tr>
<tr>
<td></td>
<td>InteractionDetail.role1</td>
<td>InteractionDetail.type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type</td>
</tr>
</tbody>
</table>

chat  http://chat.intermine.org (Our public support channel on discord)

telephone  +44 (0)1223 760 262

fax  +44 (0)1223 760 241 (please clearly indicate: InterMine - Micklem lab)

post

InterMine
Systems Biology Centre
Tennis Court Road
CAMBRIDGE CB2 1QR
United Kingdom

in person (please contact us first so we know to expect you)  http://map.cam.ac.uk/Cambridge+Systems+Biology+Centre
Protein Domains

<table>
<thead>
<tr>
<th>class</th>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProteinDomain</td>
<td>proteins</td>
<td>proteinDomainRegions</td>
</tr>
<tr>
<td></td>
<td>proteinDomains</td>
<td>proteinDomainRegions</td>
</tr>
<tr>
<td>Protein</td>
<td>–</td>
<td>start</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>identifier</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>database</td>
</tr>
<tr>
<td>ProteinDomainRegion</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

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.

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
```

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.postgresql.ds.PGSimpleDataSource
```

```
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.postgresql.ds.PGSimpleDataSource
```

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.

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.

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

```xml
<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

Updated to latest version of Sequence Ontology, 2.5

<table>
<thead>
<tr>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment.text</td>
<td>Comment.description</td>
</tr>
<tr>
<td>Gene.ncbiGeneNumber</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>Gene.description</td>
</tr>
<tr>
<td>–</td>
<td>Gene.briefDescription</td>
</tr>
</tbody>
</table>

**Interactions**

<table>
<thead>
<tr>
<th>class</th>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>gene</td>
<td>gene1</td>
</tr>
<tr>
<td></td>
<td>interactingGenes</td>
<td>gene2</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>details.type</td>
</tr>
<tr>
<td></td>
<td>role</td>
<td>details.role1</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>details.role2</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>details.name</td>
</tr>
<tr>
<td></td>
<td>shortName</td>
<td>–</td>
</tr>
<tr>
<td>InteractionRegion</td>
<td>primaryIdentifier</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>–</td>
</tr>
</tbody>
</table>

**Gene Ontology**

<table>
<thead>
<tr>
<th>class</th>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOAnnotation</td>
<td>withText</td>
<td>evidence.withText</td>
</tr>
<tr>
<td></td>
<td>with</td>
<td>evidence.with</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>annotationExtension</td>
</tr>
<tr>
<td>OntologyTerm</td>
<td>–</td>
<td>crossReferences 25</td>
</tr>
</tbody>
</table>

**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 interoperability with other InterMines), you should use the Entrez Gene ID resolver:

2. Unzip the file
3. Add the path to properties file:

25 used for Uberon
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.

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

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
- **modMine** - a data warehouse including a repository for modENCODE project fly and worm data, alongside analysis tools
- **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
- **metabolicMine** - a data warehouse targeted at the metabolic disease community, containing relevant datasets from rat, mouse and human
- **TargetMine** - a data warehouse for candidate gene prioritisation and drug target discovery, developed at NIBIO, Japan
- **mitominer** - a data warehouse of mitochondrial proteomics data for a range of organisms

More information:

InterMine Features

InterMine is a powerful open source data warehouse system, created specifically for integrating and analysing complex biological data. Benefiting from over a decade of data warehousing experience and input from a wide range of research collaborators, InterMine is still in active development, and is used by a number of major model organism databases among others. InterMine features include:
Sophisticated data integration facilities

While a core biological model based on the Sequence Ontology is provided, the data model is flexible and extensible – new data types can be added easily by editing an XML file. A range of data parsers is provided to facilitate the data loading, and a number of consistency checks after the database build ensure that the data has been integrated correctly. Sophisticated identifier resolution ensures that all data identifiers are correctly updated to their most current form.

Fast, flexible querying

The sophisticated query optimisation means that users can construct and perform a wide range of queries across the data model, while retaining good query speed. The query optimisation method is constructed around the use of precomputed tables, meaning that the data schema does not need to be denormalized in order to speed up query time. The system is also fast enough to deal with large quantities of data - the HumanMine database contains over 50 million objects, and its size with precomputed tables is 200 GB, with PhytoMine being much larger, containing 2 billion objects and almost 1500GB.

User-friendly web interface and analysis tools

The web application is included with the InterMine package, and is an accessible starting point for first time users. It contains a number of features focused around list analysis (a common need in biology) including graphical data displayers and tools that automatically calculate a set of enrichment statistics. It also includes report pages, interactive results tables, saved template queries, a regions search tool and a query builder. This setup makes it possible to browse and explore data without any programming knowledge. Users can save their data and queries in a private workspace.

Extensive set of APIs and web tools

InterMine can be accessed programmatically, and we provide client libraries for five commonly used programming languages (Python, Perl, Ruby, Java, JavaScript). This enables bioinformatician users to access InterMine functionality without using the web application and to query data from a number of different InterMine instances using a single script, or as part of an automated workflow. It also enables the easy embedding of InterMine analysis tools into external websites, as well as the development of external applications that access InterMine data.

Highly developed and extensible system

InterMine has been in development for over 10 years, and during this time, based on user demand, we have introduced a large number of features. These range from faceted filtering options and enabling Boolean logic and set operations, to table sorting and filtering, a range of standardised export options, integration of other tools such as Cytoscape, and enabling embedding of individual analysis tools as part of external websites. With funding secured for a further 5 years, we plan to continue adding features to InterMine. Furthermore, the open source, extensible framework means InterMine is also open to other developers to build upon.

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)
How to cite us

If you use the InterMine framework in your research, we would appreciate it if you cite the following publication:

• InterMine: a flexible data warehouse system for the integration and analysis of heterogeneous biological data. Smith RN et al. (2012). Bioinformatics, in press.

Individual data warehouses also have specific publications associated with them. To cite those in a paper, where available, please use the individual publication associated with the data warehouse in question:


Legal

All InterMine code is freely available under the open source LGPL license.

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.

Release Notes

See https://github.com/intermine/intermine/releases

Vacancies

Get the Software

/about/jobs/software-developer-javascript
/about/jobs/index

Get Involved

So You Want to Develop for InterMine

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. This document is normative.

There is no distinction between the processes that developers should follow internally or externally - all code contributions, whether from core team members or outside contributors, 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 any of the special branches. 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 recyle everything - then we would do the following:

1. Checkout the current head of dev from upstream.
2. Branch \textit{dev}, naming the branch something descriptive like \texttt{womblier}.

3. Checkout the new branch.


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 \texttt{you/womblier} be merged into \texttt{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 \texttt{intermine/dev} into \texttt{you/womblier}, especially if development is taking a long time. These merges should probably be \texttt{rebase} merges.

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

\textbf{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 \texttt{master} and \texttt{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 (\texttt{master} or \texttt{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 \texttt{intermine/dev} and merge the feature branch into it.

6. Push \texttt{dev} to the \texttt{intermine} repo.

[intermine repo]: https://github.com/intermine/intermine

\textbf{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.
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.

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.
Indices

- genindex
- search
Symbols

Symbols

A

about us, 275
acceptance tests, 114
Amazon, 49
anatomy ontology, 83
ant, 7, 19, 36, 49
ANT_OPTS, 8
antlib-int.xml, 265
api, 258
api loader, 256
Apples, 19
apples, 18
apps, 223, 227, 228, 252
aspects, 189
attribute links, 186
authentication, 186, 209
autocomplete, 186, 209
AWS, 49

B

bag-queries, 147
bagqueryrunner, 147
Benjamini Hochberg, 155, 214
BioGRID, 65
biopax, 67
bioseg, 15, 57
blog, 266, 272
blue genes, 121
Bonferroni, 155, 214
build failed, 111
build-db, 36, 101
building database, 101

C

Cambridge, 266, 272
careers, 275
catalina.out, 265
categories, 189
CDN, 205
chado, 77
chat, 266, 272
chromosome location, 52
cite, 275
class descriptions, 191
class keys, 187
classpath, 265
clients, 209
cloud, 49
code generation, 209
complexes, 66
components, 70
cfg, 125
Conflicting values for field error, 106
connection pool, 50
contact, 266, 272
contact form, 178
Content Delivery Network, 205
converter templates, 147
cookies, 274
create account, 178
create-attribute-indexes, 108
create-autocomplete-index, 108
create-bioseg-location-index, 57, 108
create-chromosome-locations-and-lengths, 108
create-gene-flanking-features, 108
create-intergenic-region-features, 108
create-intron-features, 108
create-overlap-view, 57, 108
create-references, 108
create-search-index, 108
creating a database, 36
cross reference links, 186
custom data source, 90
Cytoscape, 193

D

Daction, 101
DAS, 195
das, 198
data, 120
data categories, 189
data downloading, 60
data integration, 36, 102
data integrity, 113, 114
data labels, 57
data loading speed, 117
data model, 52, 55, 103
data model overview, 52
data page, 189
data set, 52
data sets, 84
data source, 52
data sources, 84
data tab, 189
database, 50
database building, 111
database properties, 178
database speed, 120
Debian, 5
debugging, 111
default-template-queries.xml, 204
deletions, 73
deploy URL, 178
developers, 262
disease data, 82
displayer examples, 130
do-sources, 108
Drelease, 178
dropdowns, 165
Dsourc, 101

E
EBI, 66
Eclipse, 19
email, 178, 262, 266, 272
embedding, 150, 209, 223, 227, 228, 252, 256, 258
emptySessionPath, 17
enrichment widgets, 155, 214
Ensembl, 78
Ensembl Compara, 64
entrez-organism, 83
error messages, 265
EsyN, 194
export, 178
exporters, 187

F
FASTA, 36, 70, 79
favicon, 173
fax, 266, 272
featured lists, 125
feedback, 178
FlyBase, 77, 82
footer, 173
forgot password, 178
funding, 178

G
Galaxy, 195
GBrowse, 195, 198
gene ontology, 61
genes, 67
genome browser, 195
genome coordinates, 57
genomic region search, 175
GFF3, 36, 81
git, 8, 19, 20
GMail, 186, 204
GO, 61, 70
Google, 186, 201, 204
Google analytics, 198
GVF, 72

H
hardware, 5, 117
header, 173
header links, 186
help, 191, 262, 265
help location, 178
hide, 169
Hikari, 50
hiring, 275
Holm-Bonferroni, 155, 214
home page, 125
Homologene, 62
homologues, 65
Hypergeometric Distribution, 155, 214

I
ID Map, 111
id resolver, 65
IDE, 19
identifiers, 82, 99
insertions, 73
IntAct, 66
Intellij, 19
interaction viewer, 194
interactions, 66, 193
interactions widget, 193
InterMine, 271
InterMine items XML, 82
intermine.log, 265
InterPro, 71
IQL, 265
isoforms, 70
J
Java, 8, 19, 209
Java Items API, 93
JAVA_OPTS, 17, 265
JavaScript, 209, 258
javascript embedding, 150, 209, 223, 227, 228, 252, 258
javascript loader, 256
JBrowse, 195, 198
jobs, 275
jwt, 186

K
KEGG, 67
keys, 36, 265
keyword search, 171
keyword search examples, 186
keywords, 70

L
LATIN-9, 13
layout, 173
LGPL, 275
link outs, 186
links, 192
list analysis page, 150
list analysis page widgets, 150, 209
list upgrade, 150
list upload, 147
list upload classes, 187
list upload examples, 186
list widgets, 150, 209
lists, 146, 163
literature, 72
LOG, 265
logo, 173
logs, 36, 265
look & feel, 173
LOOKUP, 147

M
Mac, 18, 19
mailing list, 262
make_mine, 36
malariamine, 36, 49
meta description, 186
meta keywords, 186
MGI, 82
mine links, 192
mine properties, 178
model, 55
model merging, 103
model properties, 178
multiple mines, 178
Multiple Test Correction, 155, 214
myid.net, 204

N
NCBI, 72, 83
network viewer, 193, 194

O
oauth, 186
oauth2, 186
OBO, 61
old identifiers, 99
OMIM, 82
Open ID, 204
openid, 186, 204
optimisation, 120
optional constraints, 165
Oracle, 87
organism, 52
organism name, 52
organisms, 83
OrthoDB, 62
orthologues, 65
out of memory error, 17
OutOfMemoryError, 8, 265
overlappingFeatures, 57
overlaps, 57

P
Panther, 63
paralogues, 65
pathways, 67
PDB, 70
performance, 117, 205
Perl, 19, 209
Perl installation, 11
Perl Items API, 95
PermGen, 265
phone, 266, 272
PNG, 193
popular templates, 125
portal welcome message, 186
postgres, 117
PostgreSQL, 13, 19
precomputes, 120
primary keys, 36, 105
priority configuration, 106
priority conflicts, 36
privacy, 274
privacy policy, 275
project title, 178
project XML, 36
project_build, 49
project_build script, 101
Protein Data Bank, 70
protein domains, 70, 71
protein features, 70
protein structures, 70
proteins, 67, 70, 71
provenance, 84
ProxyReference, 111
PSI, 66
PSQLException, 265
publications, 36, 72
PubMed, 71
Python, 209

Q
query results, 168, 256
query speed, 120
querybuilder, 169
quick search, 171

R
Reactome, 67
region search, 57, 175
release notes, 275
release version, 178
report displayers, 129, 130, 187
report page, 125, 129, 130, 135, 145
report widgets, 223
report widgets service, 223
resolvers, 99
REST, 209
RGD, 82
robots.txt, 201
RSS, 125
Ruby, 209
running a build, 101

S
search, 171
search engines, 201
SEO, 201
sequence features, 81
sequence ontology, 83
sequences, 79
session error, 17
sessionCookiePath, 17
SGD, 82
showProperties, 265
SIF, 193
SMTP, 178
SNPs, 72, 73
SNVs, 73
SO, 83
software dependencies, 19
solaris, 5
speed, 117
SQL_ASCII, 13
strand, 52
subtitle, 178
summarise-objectstore, 108
summary, 169
superuser, 178, 203
SVG, 193
system requirements, 19

T
tabs, 173
tagging, 203
take a tour link, 191
taxonomy ID, 83
template comparison, 113
template queries, 165
test model, 43
tests, 46
themes, 173
title, 178
tokens, 209
tomcat, 17, 19
too many clients error, 265
tracking Mine usage, 200
transfer-sequences, 108
TreeFam, 62
tutorial, 36, 49, 50
twitter, 266, 272

U
Uberon, 83
UniProt, 36, 67, 70
unit tests, 46
user-agent, 201
userprofile, 36, 204
UTF-8, 13

V
vacancies, 275
variant file format, 73
variation, 72
vcf, 73

W
web properties, 186
web services, 209
webapp admin, 203
webapp properties, 178
webapp speed, 205
webapp tables, 135
webconfig-model.xml, 187
widgets, 150, 155, 187, 209, 214, 223, 227, 228, 252, 258
WormBase, 77
writing a custom data source, 90

X
XGMML, 193

Y
Yahoo, 201, 204

Z
ZFIN, 82