InterMine Documentation

Release

InterMine

January 25, 2016

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

CHAPTER 1

Contents

1.1 System Requirements

1.1.1 Hardware

Recommendations

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

Database servers

- 8 cores
- 32 GB RAM
- ~2TB usable storage (SAS disks are faster than SATA but more expensive)
  - 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

A less expensive option:

- 4 cores
- 16GB RAM
- ~2TB usable storage
  - RAID 10
  - hardware RAID controller with a battery backed cache (gives faster write speeds)
- Linux/Unix capable of running Java and PostgreSQL

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

Web servers

- 4 cores
- 8GB RAM (or more)
- ~200GB local disks

A less expensive option is to run Tomcat on the database machine and not buy separate web servers, we do this for modENCODE and performance is fine.

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:
  - CentOS
  - 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.

Database sizes/ disk space

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

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

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

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

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

Related topics:

Solaris Installation guide

Installation notes

Update postgres.conf

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

Improvements for COPY

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

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

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

Run analyse

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

1.1.2 Software

InterMine makes use of a variety of freely available software packages. All of the software listed below is required.

Software	Minimum Version	Purpose
Git	1.7	check out and update source code
Java SDK	6.0	build and use InterMine
Ant	1.8	invokes the InterMine build
Tomcat	6.0.29	website
PostgreSQL	8.3	database
Perl	5.8.8	run build scripts

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:

```
$ ant build-db
```

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

```
$ 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:

```
$ 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

Target	Purpose
clean	
clean-all	clean this project and all dependencies
compile	
jar	produce jar/war
generate	create generated code
action	do the action for this project, if any
javadoc	build javadoc to build/javadoc

Tests

Target	Purpose
default	will produce a jar and run tests
test	
test-all	test all dependencies as well as this project

To run a single test class:

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

DBmodel

Target	Purpose
build-db	build the objectstore

Webapp

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

Test-model

Target	Purpose
insert-data-unittest	insert test data into database

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

```
$ 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:SoftRefLRU
```

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.

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 PyPi in Python], Yum/Apt repositories in Linux, or even Apple's App Store). If you have Perl you have CPAN. (To check type *cpan* in a terminal).

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

```
$ cpan
```

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

```
$ o conf
```

When you first run cpan should run:

```
$ o conf init
```

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

To change a setting manually, type:

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

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

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

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

```
$ o conf commit
```

To install modules, type:

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

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

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

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

DEB Packages

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

The procedure is the same as for any other package:

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

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

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

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 four "distributions" of modules you may want to install:

- InterMine-TypeLibrary
- InterMine-Model (depends on InterMine::TypeLibrary)
- 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::Roles::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)
- Glob
- Text::Glob
- for the download scripts:
- Log::Handler
- DateTime
- Module::Find
- Web::Scraper
- Ouch
- Number::Format
- PerlIO::gzip
- Perl6::Junction
- for generating InterMine Items XML:
- Getopt::Std
- Log::Handler;
- Digest::MD5

PostgreSQL

Installing PostgreSQL

http://www.postgresql.org/download has packages for most systems that set up everything for you. You should use at least version 8.3, as we have recently removed workarounds for some bugs in previous versions. Postgres 8.4 is the default for most package managers and has better performance.

- Fedora/CentOS: http://wiki.openscg.com/index.php/PostgreSQL_RPM_Installation
- Debian/Ubuntu: *sudo apt-get postgresql-8.4*
- Mac: http://www.postgresql.org/download/macosx. We've had the most success with MacPorts.
- Solaris: http://wiki.postgresql.org/wiki/Detailed_installation_guides#Solaris

After installation, you need to update *postgresql.conf*

- This file is usually located in */etc/postgres/8.4*:
- If you are going to install Postgres 9.x:
  - It's not easy to change the default encoding to SQL_ASCII anymore, so you should do this before creating any databases.
  - There are special instructions for installing BioSeg

Required Configuration

listen_addresses	'*'
tcpip_socket [1]	true
port	5432

Recommended Configuration for optimum performance

shared_buffers	Set to around 150MB
temp_buffers	Set to around 80MB
work_mem	Set to around 500MB but not more than 1/10 of available RAM
maintenance_work_mem	Set to around 3000MB but not more than 1/5 of available RAM
default_statistics_target	Set to around 250
random_page_cost	Set to around 2.0, rather than 4.0
effective_cache_size	Set to about 2/3 the amount of RAM in the computer
geqo_threshold	Set to 14
from_collapse_limit	Set to 14
join_collapse_limit	Set to 14
max_locks_per_transaction	Set to 640

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
```

Note that changing some settings requires stopping/starting postgres, restart has no effect.

You also need to install the *bioseg* data type, and the *contrib btree_gist* plug-in, as described in BioSeg.

[1] not needed in recent releases

Character Set Encoding We recommend using either *SQL_ASCII* or *UTF-8*. Theoretically, we should be using *UTF-8*, which is more correct, however its performance is rather poor, so we use *SQL_ASCII*.

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

If you set Postgres 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 Postgres 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.

BioSeg

Prerequisites

Contrib software

debian	apt-get install postgresql-contrib-8.4
fedora	yum install postgresql-contrib

Header files

debian	apt-get install postgresql-server-dev-8.4
fedora	yum install postgresql-devel

pg_config

debian	apt-get install libpq-dev

Installation

1. Download the bioseg tar from http://www.bioinformatics.org/bioseg/wiki/.
2. Untar the file, change to the created directory and run these commands

```
$ 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

debian	/usr/share/postgresql/8.4/contrib
fedora	/usr/share/pgsql/contrib

For each database, type:

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

Gist We also need to create the default gist operators too, in order to have normal types in multi-column indexes.

Postgres 8.x users For each database, type:

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

Postgres 9.x users See 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. 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
```

1. Use the *startup.sh* and *shutdown.sh* scripts in *apache-tomcat-x.y.z/bin/* to start and stop Tomcat.
   (a) You can run it as your own user for development purposes, you should never run tomcat as *root*.
2. Set up the manager user as below.

See Tomcat's installation instructions - http://tomcat.apache.org/tomcat-6.0-doc/setup.html.

After Installation

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

Tomcat 6.0.xx

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

Tomcat 7.0.xx

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

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

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

```
$ apt-get install tomcat6-admin
```

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

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

Without this, permalinks may break.

Tomcat 7.0 Starting from InterMine 1.1, you can now deploy your mine using Tomcat 7.0

Update your config files:

Add a new property (default is 6.0.xx)

```
# in MINE.properties
webapp.tomcat.version=7
```

in *$TOMCAT/conf/tomcat-users.xml* set:

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

Add the property in startup.sh:

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

Tomcat 7.0 has improved the enforcement of the Expression Language rules and doesn't allow to use java key words. The flag makes Tomcat 7 more permissive.

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

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

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

Stopping Tomcat To stop Tomcat, run this command:

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

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

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

Common Errors

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

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

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

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

Tomcat 6.0 Add *emptySessionPath* to *$TOMCAT/conf/server.xml*

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443"
  emptySessionPath="true" />

<Connector port="8009" protocol="AJP/1.3"
  redirectPort="8443"
  emptySessionPath="true"  />
```

Tomcat 6.0 HTTP connector documentation - mentions potential security hole

Tomcat 7.0 Add these 2 attributes to *$TOMCAT/conf/conf/context.xml*

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

Tomcat 7.0 context documentation

Mac Installation Notes

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

Installing Tomcat

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

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

Extract this file with:

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

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

- *bin* - scripts to startup and shutdown tomcat
- *logs* - error logs are written here

- *webapps* - the directory web applications are deployed to
- *conf* - configuration files

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

Start up tomcat by running:

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

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

Installing Eclipse

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

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

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

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

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

Installing Postgres

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

1.2 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.

1.3 Get started

1.3.1 Tutorial

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

- Load some real data sets for Malaria (''P. falciparum'')
- Learn about 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

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 anyting 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 a 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:

```
$ cd malariamine
$ ls
```

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:

```
$ 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:

```
$ cp ../bio/tutorial/malariamine/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:

```
$ cd
$ cp git/intermine/bio/tutorial/malariamine/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/\/home\/username/g' project.xml
```

For example, the *uniprot-malaria* source:

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

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

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

Properties file

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

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

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

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

```
$ cd
$ cp git/intermine/bio/tutorial/malariamine/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
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:

```
$ 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"/>
  <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 so_term_list.txt. The build system creates classes corresponding to the Sequence Ontology terms:

```
$ less resources/so_term_list.txt
```

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:

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

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

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

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

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

This has also created necessary indexes on the tables:

```
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:

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

Loading Data

We are running 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:

```
$ 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, this page should help diagnose the problem: CommonErrors.

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:

```
[ant] load:
[ant]      [echo]
[ant]      [echo]       Loading uniprot-malaria (uniprot) tgt items into production DB
[ant]      [echo]
```

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 SequenceFeatures (representing any genome feature) both Genes and Exons will be returned automatically.

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

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

```
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 use 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:

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

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

col 2: "source" the database or algorithm that provided the feature

col 3: "type" a valid 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.

Other *gff3* properties can be congfigured 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 *primaryIdentifier* and *symbol*/*secondaryIdentifier* in the features created. This is to match the identifiers from UniProt, it is quite a common issue when integrating from multiple data sources.

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

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/dataconvers
```

Loading GFF3 data

Now load the *malaria-gff* source by running this command in *malariamine/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 *dbmodel* 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 malariamine
malariamine#  select class, count(*) from intermineobject group by class;
```

FASTA files

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

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

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

fasta.className = org.flymine.model.genomic.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

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

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

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

```
malariamine=# select id, primaryidentifier, secondaryidentifier, symbol, length , chromosomeid, chro
    id    | primaryidentifier | secondaryidentifier | symbol | length | chromosomeid | chromosomeloca
----------+-------------------+---------------------+--------+--------+--------------+---------------
83000626 | PFL1385c          | gene.33449          | ABRA   |   2232 |     84017653 |              84
(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.flymine.model.genomic.Or
(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.

Dealing with conflicts

It is possible that two different sources could have a value for the same field, in this case we need to define which source should take precedence.

Try loading data from the *interpro* source which will cause a conflict with *uniprot-malaria*. UniProt loads a list of protein domains for each protein but doesn't include the full names and descriptions of ProteinDomain objects, the *interpro* source adds this information:

```
$ ant -Dsource=interpro
```

You will see a message that includes the text:

```
Conflicting values for field ProteinDomain.shortName between uniprot-malaria (value "NAD-dep_Gly3P_DH
```

Slightly different values have been provided for the *shortName* of a particular *ProteinDomain*. In this case we want to take the value from *interpro*. Edit *dbmodel/resources/genomic_priorities.properties* and add the line:

```
ProteinDomain.shortName = interpro, uniprot-malaria
```

Now run try to load *interpro* again:

```
$ ant -Dsource=interpro
```

The *tracker* table

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

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

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

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

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

Updating Organism and Publication Information

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

Fetching organism details

You will have noticed that in previous sources and in *project.xml* we have referred to organisms by their NCBI Taxonomy id. These are numerical ids assigned to each species 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:

```
$ 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:

```
$ 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 sources InterMine sources load publications:

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

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

```
$ 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:

```
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-procesing

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.

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/. The username and password required to access the manager are *webapp.manager* and *webapp.password* as specified in malariamine.properties.

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 to view your webapp. The path to your webapp is the *webapp.path* value set in malariamine.properties.

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.

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 [source:trunk/bio/tutorial/malariamine/malaria-data.tar.gz bio/tutorial/malariamine/malaria-data.tar.gz]. Copy this file to your local directory and extract from the archive.

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

Edit the project.xml file so that all occurances of ''DATA_DIR" point to the 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.

gene_ontology The Gene Ontology structure. Downloaded from http://www.geneontology.org/

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

1.3.2 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.

```
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 testmodel-webapp
$ createdb testmodel-webapp-userprofile
```

Initialise the two databases:

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

Build the testmodel web application and release to the running tomcat:

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

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.

1.3.3 InterMine Tests

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.

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.

```
$ createdb unittest
$ createdb truncunittest
$ createdb fulldatatest
$ createdb flatmodetest
$ createdb notxmltest
```

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:

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

Update properties file

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

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

# common properties

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

# testing properties

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

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

Build the databases

Build database tables automatically generated from the bio model by running the following in *bio/test-all/dbmodel*:

```
$ ant clean build-db
```

Run the tests

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

```
$ 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.

```
# 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.

1.4 Data Model

1.4.1 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 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.

1.4.2 Using Class and Field Labels

The InterMine webapp, and to a limited extent web services, supports the use of labels for classes and fields. Unlabelled classes and fields are formatted for enhanced legibility.

The current system for determining a label is as follows:

1. If the class or field has a pre-set label, that is used
2. Otherwise the class or field name is

1. Split from its camel case parts as specified in Apache Commons StringUtils

2. Each part is given an initial upper-case
3. The parts are then joined by spaces

Handling paths is similar, except that the dots (".") between class and field names are replaced by right angle-brackets (">").

Examples

Before	After
ChromosomeLocation	Chromosome Location
shortName	Short Name
Organism	Organism
name	Name
Organism.shortName	Organism > Short Name

Well named fields and classes thus do not need explicit labelling.

Labels can be configured however in two ways, in order of precedence: #. Classes and fields can be configured individually. This configuration respects inheritance, and subclasses automatically inherit the field labels of their parents. #. Translation tables can be set up for classes and fields. These are for cases where *ALL* classes/fields with a certain name should be relabelled. Examples are *url -> URL*, which would otherwise be rendered as ''Url''. This is especially useful for acroynms.

Configuring classes and fields individually

To apply individual configuration, the file *webconfig-model.xml* needs to be edited, and a *label* attribute added to items you want to configure. eg:

```
<class className="org.intermine.model.bio.Allele" label="SOME CLASS LABEL">
    <fields>
      <fieldconfig fieldExpr="primaryIdentifier" label="SOME FIELD LABEL"/>
      <fieldconfig fieldExpr="symbol"/>
      <fieldconfig fieldExpr="alleleClass"/>
      <fieldconfig fieldExpr="organism.name" label="Organism"/>
    </fields>
</class>
```

This is most helpful in the case of compound field-expressions ("organism.name"), which can this be configured to display as just a single expression.

Configuring classes and fields globally

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, and exception will be thrown on initialisation, and your webapp will not start.

Using these labels in your webapp

A new tag library is available to help with labelling. Add the following to the top of any jsp you write that you want to use labels in:

```
<%@ taglib uri="/WEB-INF/functions.tld" prefix="imf" %>
```

This library provides five functions, which expose static methods from the org.intermine.web.logic.WebUtil class:

formatPath(Path p, WebConfig wcf)

> **This function produces a fully configured string from an arbitrarily long path. eg:** *<c:out value="${imf:formatColumnName(path, WEBCONFIG)}"/>*

formatPathStr(String s, InterMineAPI api, Webconfig wcf)

> **This function produces a fully configured string from an arbitrarily long path, where that path is represented as a string.**
> *<c:out value="${imf:formatColumnName(pathString, INTERMINE_API, WEBCONFIG)}"/>*

formatField(Path p, Webconfig wcf)

> **This function produces a fully configured field name from the last field of an arbitrarily long path. eg:**
> *<c:out value="${imf:formatField(path, WEBCONFIG)}"/>*

formatFieldStr(String s, InterMineAPI api, Webconfig wcf)

> **This function produces a fully configured field name from the last field of an arbitrarily long path, where that path is repr**
> *<c:out value="${imf:formatFieldStr(pathString, INTERMINE_API, WEBCONFIG)}"/>*

formatFieldChain(String s, InterMineAPI api, Webconfig wcf)

> **This function produces a string of fully configured field names from all the fields in an arbitrarily long path, where that pa**
> *<c:out value="${imf:formatFieldStr(pathString, INTERMINE_API, WEBCONFIG)}"/>*

The values *INTERMINE_API* and *WEBCONFIG* are automatically available within jsps at all times.

While it is possible to call the formatting methods of WebUtil directly from Java controllers, it is not advisable, from design principles, to do so. Labels are an aspect of presentation (the view) and thus not the responsibility of Java classes (the controllers). The only justifiable place to call presentation methods from is in action classes that directly return data to the user, eg. in webservices and ajax calls.

Using Labels in JavaScript

Pages in the InterMine webapp have a variable in the global scope named *$MODEL_TRANSLATION_TABLE*. This contains information on how all classes and their fields should be displayed.

To access its information, for classes:

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

And for fields of this class:

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

1.4.3 Querying over genomic ranges

InterMine includes functionality for querying features with overlapping genome coordinates. We have a custom *BIOSEG* index for postgres 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 *BIOSEG* index to find other features that overlap it.

In modMine (the InterMine for the modENCODE project] we also create *GeneFlankingRegion* features to represent specific distances upstream and downstream of genes to query for genes that are nearby other features.

To use the overlapping queries in InterMine you need to:

Install BIOSEG

BIOSEG is a custom index type that you need to add to postgres. See BioSeg for details

Create the BIOSEG index

You need to create the bioseg index on the location table in your production database. This is done simply by adding the *create-bioseg-location-index* post-process step to your *project.xml* file:

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

Create the *overlappingFeatures* view

Now create the view in 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. Just add the *create-overlap-view* post-process step, it needs to be after *create-bioseg-location-index*.

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

See FlyMine's project.xml for an example of including these steps.

Now any queries on the *overlappingFeatures* collections will use this view and bioseg.

1.5 Database

1.5.1 Data Download Scripts

The DataDownloader system uses a plugin architecture to make it more straightforward to download data from arbitrary sources, and to add new sources to the system

Location

The system is a package located 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 liblog-handler-perl liblog-
```

Other perl modules need to be installed via CPAN:

```
$ cpan
cpan[1]> install MooseX::ABC
cpan[2]> install MooseX::FileAttribute
```

Data Source Configuration

To learn how to configure data sources of your mine, look here for examples:

bio/scripts/DataDownloader/config

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

Running

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

```
perl 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*:

```
package DataDownloader::Source::FlyAnatomyOntology;

use Moose;
extends 'DataDownloader::Source::ABC';

use constant {
    TITLE  => 'Fly Anatomy Ontology',
    DESCRIPTION => "Drosophila Anatomy ontology from FlyBase",
    SOURCE_LINK => "http://www.flybase.net/",
    SOURCE_DIR => 'ontologies/fly-anatomy',
    SOURCES => [{
        FILE   => 'fly_anatomy.obo',
        SERVER => 'http://obo.cvs.sourceforge.net/*checkout*/obo/obo/ontology/anatomy/gross_anatomy/a
    }],
};

1;
```

This source fully inherits the behaviour of the 'DataDownloader::Source::ABC' abstract class, and only adds configuration. In this case, it defines a set of constants that describe this source:

- 'TITLE': The human readable name of the source shown in log messages.
- 'DESCRIPTION': A Longer description of the data that appears in a version file.
- 'SOURCE_LINK': A link to the origin of the material that appears in the version file.
- 'SOURCE_DIR': The sub-directory under the 'data_dir' of the constructor where the new files should be placed.

And some constants that define the data to fetch:

- 'SOURCES': Any data sources defined by this constant will automatically be added to the queue of files to download.

Each source is a hash-reference with the following keys:

- 'FILE': The name of the file on the remote server
- 'SERVER': The path to the location of the file to fetch.

Further keys that can be defined include:

- 'POSTPROCESSOR': A code-reference which will called as a method and passed the downloaded file, and the location where it should end up.

1.5.2 Data Sources

Contents

Data Source Library

This page lists the current sources available for use in InterMine. All the sources here are found 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

Currently we require that the gene_association file is sorted by the 'product' (gene/protein/etc) identifier in column 2. This is to save memory while loading. If your file isn't already sorted by column 2 you can use this command

```
$ sort -k 2,2 input_file > output_file
```

How to load the data into your mine project XML example

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

GO OBO Load the Gene Ontology term ids, names and definitions, and the relationships between terms. Should be loaded if the go-annotation source is used.

Types of data loaded GO terms

How to download the data From http://www.geneontology.org

How to load the data into your mine project XML example

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

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

Treefam

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

Download two tables:

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

Project XML

```
<source name="treefam" type="treefam">
  <property name="src.data.dir" location="/DATA/treefam"/>
  <property name="src.data.dir.includes" value="ortholog.txt.table"/>
  <property name="geneFile" value="/DATA/treefam/genes.txt.table"/>
  <property name="treefam.organisms" value="7227 6239 7165 4932"/>
  <property name="treefam.homologues" value="9606 10090 10116 7955"/>
</source>
```

- ''treefam.organisms'' - all genes from the listed organisms will be processed
- ''treefam.homologues'' (optional) - genes will *only* be loaded into the database if they are a homologue of an organism of interest

Homologene

1. Data

ftp://ftp.ncbi.nih.gov/pub/HomoloGene/current/homologene.data

2. project.xml

```
<source name="homologene" type="homologene">
  <property name="src.data.dir" location="/DATA/homologene"/>
  <property name="homologene.organisms" value="7227 9606 10090 10116 7955 6239 4932"/>
</source>
```

OrthoDB Data

ftp://cegg.unige.ch/OrthoDB5/OrthoDB5_ALL_tabtext.gz

Project XML

```
<source name="orthodb" type="orthodb">
  <property name="src.data.dir" location="/DATA/orthodb"/>
  <property name="orthodb.organisms" value="7227 9606 10090 10116 7955 6239 4932"/>
</source>
```

Panther Data

ftp://ftp.pantherdb.org/ortholog/current/RefGenomeOrthologs.tar.gz

gunzip to RefGenomeOrthologs.txt

Project XML

```
<source name="panther" type="panther">
  <property name="src.data.dir" location="/DATA/panther"/>
  <property name="panther.organisms" value="7227 9606 10090 10116 7955 6239 4932"/>
</source>
```

Ensembl Compara

Download data from BioMart

1. [http://www.biomart.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 3 orthologs):* to open that section of the form
4. select on the Gene ID for the organism of interest, eg. Drosophila Ensembl Gene ID

1. Run query

1. select the *[Results]* button at the top of the page
2. create *TSV* file, check box next to *Unique results only*
3. when prompted, save file as *TAXONID1_TAXONID2*

Add entry to project XML file

```
<source name="ensembl-compara" type="ensembl-compara">
  <property name="src.data.dir" location="/DATA/ensembl/compara"/>
  <property name="ensemblcompara.organisms" value="7227"/>
  <property name="ensemblcompara.homologues" value="6239"/>
</source>
```

Run build

Data file Tab-delimited files should be named <TAXON ID>_<TAXON ID>, eg. 9606_10090 for a file with human genes and mouse orthologues.

Gene ID	Homologue ID
ENSG00000253023	ENSMUSG00000088328
ENSG00000238364	ENSMUSG00000088728

Download script When you have created your query, you can export the Perl script or XML so you can run the query automatically next time, eg:

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

  <Dataset name = "hsapiens_gene_ensembl" interface = "default" >
    <Filter name = "with_dmelanogaster_homolog" excluded = "0"/>
    <Attribute name = "ensembl_gene_id" />
    <Attribute name = "drosophila_ensembl_gene" />
  </Dataset>
</Query>
```

Identifiers The default rule for bio-InterMine is to put the MOD identifiers (eg. MGI:XXX or ZDB-GENE-XXX) in the primaryIdentifier field. This is tricky because some homologue sources use the Ensembl identifiers (Ensembl identifiers belong in the Gene.crossReferences collection).

To solve this problem, each homologue source uses the NCBI identifier resolver. This resolver takes the Ensembl ID and replaces it with the corresponding MOD identifier.

To use the NCBI gene resolver, see Id Resolvers which also provides more information on how ID resolvers work in InterMine.

Interactions

BioGRID Loads interactions data from BioGRID

Types of data loaded genes, proteins, interactions

How to download the data http://thebiogrid.org/downloads/archives/Release%20Archive/BIOGRID-[RELEASE VERSION].psi25.zip

How to load the data into your mine

project XML example

```
<source name="biogrid" type="biogrid">
  <property name="src.data.dir" location="/data/biogrid"/>
  <property name="src.data.dir.includes" value="*psi25.xml"/>
  <property name="biogrid.organisms" value="7227 6239 4932"/>
</source>
```

biogrid_config.properties Determines which gene identifiers are set. organisms - If none are configured, all interactions are stored.

This is what the gene looks like in biogrid

```
 <names>
         <shortLabel>CG1111</shortLabel>
 </names>
 <xref>
<primaryRef db="FLYBASE" id="FBgn001" />
```

shortLabel

To set your gene.identifier to be the shortLabel in the biogrid XML, use this config:

```
<TAXON_ID>.<GENE_IDENTIFIER_FIELD>=shortLabel
```

xref

To set your gene.identifier field to be a value from an xref entry, use this syntax:

```
<TAXON_ID>.xref.<GENE_IDENTIFIER_FIELD> = <XREF_DB_VALUE>
```

Note: xref "db" value is not case sensitive, case seems to vary from file to file.

IntAct Loads interactions data from IntAct

Types of data loaded genes, interactions

How to download the data ftp://ftp.ebi.ac.uk/pub/databases/IntAct/current/all.zip

How to load the data into your mine

project XML example

```
<source name="psi-intact" type="psi" dump="true">
  <property name="src.data.dir" location="/data/intact"/>
  <property name="intact.organisms" value="7227"/>
</source>
```

psi-intact_config.properties Determines which gene identifiers are set. organisms - If none are configured, all interactions are stored.

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 http://psidev.sourceforge.net/mi/psi-mi.obo psi-mi.obo

How to load the data into your mine project XML example

```
<source name="psi-mi-ontology" type="psi-mi-ontology">
  <property name="src.data.file" location="/data/psi/psi-mi.obo"/>
</source>
```

Pathway data sources Content:

KEGG Link genes to KEGG pathways that they operate in.

Types of data loaded genes, pathways

How to download the data http://www.genome.jp/kegg

How to load the data into your mine

project XML example

```
<source name="kegg-pathway" type="kegg-pathway">
  <property name="src.data.dir" location="/data/kegg"/>
  <property name="kegg.organisms" value="7227"/>
</source>
```

kegg_config.properties Decides which gene identifier fields are populated, mapping from organism taxonId to abbreviation. Only taxonIds specified in project.xml file are downloaded, if no taxonIds are configured, all are loaded.

Reactome

Types of data loaded genes, pathways

How to download the data http://reactome.org/download/current/biopax.zip

How to load the data into your mine project XML example

```
<source name="reactome" type="biopax">
  <property name="src.data.dir" location="/data/reactome"/>
  <property name="biopax.organisms" value="7227"/>
  <property name="biopax.datasourcename" value="Reactome"/>
  <property name="biopax.datasetname" value="Reactome data set"/>
  <property name="biopax.curated" value="false"/>
</source>
```

Proteins

UniProt

Types of data loaded genes, proteins, GO annotation, protein domains, publications, UniProt features, comments, synonyms, cross references, EC numbers, components

How to download the data This source loads data from the UniProt website here: ftp://ftp.uniprot.org/pub/databases/uniprot/current_release

The UniProt source expects the data files to be in a special format:

```
TAXONID_uniprot_sprot.xml
TAXONID_uniprot_trembl.xml
```

How to load the data into your mine

Configuration

Gene identifier fields You can specify which gene fields are assigned when UniProt data is loaded. An example entry:

```
10116.uniqueField = primaryIdentifier
10116.primaryIdentifier.dbref = RGD
10116.secondaryIdentifier.dbref = Ensembl
10116.symbol.name = primary
```

The format for the file is:

<TAXON_ID>.<IDENTIFIER_FIELD> = <VALUE>

An example A rat uniprot entry: http://www.uniprot.org/uniprot/Q923K9.xml

The second line of that configuration would set the ID value as the gene.primaryIdentifier:

```
<dbReference type="RGD" id="619834" key="33">
        <property type="gene designation" value="Acf"/>
</dbReference>
```

The third line would set this ID value as gene.secondaryIdentifier:

```
<dbReference type="Ensembl" id="ENSRNOG00000033195" key="30">
        <property type="organism name" value="Rattus norvegicus"/>
</dbReference>
```

The last line would set the value between the <name/> tags as gene.symbol:

```
<gene>
        <name type="primary">A1cf</name>
        <name type="synonym">Acf</name>
        <name type="synonym">Asp</name>
</gene>
```

The values for symbol.name can be primary, ORF or ordered locus.

Protein feature types You can also configure which protein features to load.

To load specific feature types only, specify them like so:

```
# in uniprot_config.properties
feature.types = helix
```

To load NO feature types:

```
# in uniprot_config.properties
feature.types = NONE
```

To load ALL feature types, do not specify any feature types, remove that line from this config file. Loading all feature types is the default behaviour.

Project.xml

```
<source name="uniprot" type="uniprot" >
  <property name="uniprot.organisms" value="7227 9606"/>
  <property name="src.data.dir" location="/data/uniprot"/>
  <property name="createinterpro" value="true"/>
  <property name="createGO" value="true"/>
  <property name="allowduplicates" value="false"/>
  <property name="loadfragments" value="false"/>
  <property name="loadtrembl" value="true"/>
</source>
```

property	description	default
createinterpro	if TRUE, process protein domains	false
createGO	if TRUE, process GO annotation	false
allowduplicates	if TRUE, allow proteins with duplicate sequences to be processed	false
loadfragments	if TRUE, load all proteins even if isFragment = true	false
loadtrembl	if FALSE, not load trembl data for given organisms, load sprot data only	true

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

ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/uniprot_sprot_varsplic.fasta.gz

```
<source name="uniprot-fasta" type="fasta">
  <property name="fasta.taxonId" value="7227 9606"/>
  <property name="fasta.className" value="org.intermine.model.bio.Protein"/>
  <property name="fasta.classAttribute" value="primaryAccession"/>
  <property name="fasta.dataSetTitle" value="UniProt data set"/>
  <property name="fasta.dataSourceName" value="UniProt"/>
  <property name="src.data.dir" location="/data/uniprot/current"/>
  <property name="fasta.includes" value="uniprot_sprot_varsplic.fasta"/>
  <property name="fasta.sequenceType" value="protein" />
  <property name="fasta.loaderClassName" value="org.intermine.bio.dataconversion.UniProtFastaLoaderTa
</source>
```

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

ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/docs

```
<source name="uniprot-keywords" type="uniprot-keywords">
  <property name="src.data.dir" location="/data/uniprot/current"/>
  <property name="src.data.dir.includes" value="keywlist.xml"/>
</source>
```

Protein Data Bank

Types of data loaded proteins, protein structures

How to load the data into your mine project XML example

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

InterPro Loads the name and description for protein domains. The protein to domain relationship is loaded via the UniProt converter.

Types of data loaded protein domains

How to download the data http://www.ebi.ac.uk/interpro/interpro.xml.gz

How to load the data into your mine project XML example

```
<source name="interpro" type="interpro">
  <property name="src.data.dir" location="/data/interpro"/>
</source>
```

Publications

PubMed Data from pubmed. entire file is downloaded, only taxon IDs set in project.xml will be loaded. if nothing configured, processes all entries.

Types of data loaded genes, publications

How to download the data files

- ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene2pubmed.gz
- ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene_info.gz

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 coverter 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

```
<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

```
<source name="pubmed-gene" type="pubmed-gene">
  <property name="src.data.dir" location="DATA/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 publciation records already in the database are updated.

How to download the data Data is fetched from the NCBI web site for publication records already in the InterMine database.

How to load the data into your mine project XML example

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

Chado We have developed an InterMine data source that can use a GMOD Chado database as a source for an InterMine warehouse. The eventual aim is to allow import of any Chado database with some configuration. This will provide a web environment to perform rapid, complex queries on Chado databases with minimal development effort.

Converter The converter for this source is the *ChadoDBConverter* class. This class controls which *ChadoProcessors* are run. A *ChadoProcessor* class corresponds to a chado module. For example, the sequence module is processed by the *SequenceProcessor* and the stock module is processed by the *StockProcessor*.

Chado tables The *chado-db* source is able to integrate objects from a Chado database. Currently only tables from the *Chado sequence module* and *Chado stock modules* are read.

These tables are queried from the chado database:

feature used to create objects in the ObjectStore

- The default configuration only supports features that have a Sequence Ontology type (eg. *gene*, *exon*, *chromosome*)
- Each new feature in InterMine will be a sub-class of *SequenceFeature*.

featureloc used to create *Location* objects to set *chromosomeLocation* reference in each *SequenceFeature*

feature_relationship used to find *part_of* relationships between features

- this information is used to create parent-child references and collections
- examples include setting the *transcripts* collection in the *Exon* objects and the *gene* reference in the *Transcript* class.

dbxref* and *feature_dbxref used to create *Synonym* objects for external identifiers of features

- the *Synonym's will be added to the 'synonyms* collection of the relevant *SequenceFeature*

featureprop used to set fields in features based on properties

- an example from the FlyBase database: the *SequenceFeature.cytoLocation* field is set using the *cyto_range* feature_prop

synonym* and *feature_synonym used to create extra *Synonym* objects for *chado* synonyms and to set fields in features

- the *Synonym's will be added to the 'synonyms* collection of the relevant *SequenceFeature*

cvterm* and *feature_cvterm used to set fields in features and to create synonyms based on CV terms

pub*, *feature_pub* and *db used to set the *publications* collection in the new *SequenceFeature* objects.

Additionally, the *StockProcessor* class reads the tables from the chado stock module, eg. stockcollection, stock, stock_genotype.

Default configuration The default configuration of *ChadoDBConverter* is to query the *feature* table to only a limited list of types. The list can be changed by sub-classing the *ChadoDBConverter* class and overriding the *getFeatureList()* method. The *featureloc*, *feature_relationship* and *pub* tables will then be queried to create locations, parent-child relationships and publications (respectively).

Converter configuration Sub-classes can control how the Chado tables are used by overriding the *getConfig()* method and returning a configuration map.

Source configuration Example source configuration for reading from the ''C.elegans" Chado database:

```
<source name="chado-db-wormbase-c_elegans" type="chado-db" dump="true">
  <property name="source.db.name" value="wormbase"/>
  <property name="genus" value="Caenorhabditis"/>
  <property name="species" value="elegans"/>
  <property name="taxonId" value="6239"/>
  <property name="dataSourceName" value="WormBase"/>
  <property name="dataSetTitle" value="WormBase C.elegans data set"/>
</source>
```

Sub-classing the converter The processor classes can be sub-classed to allow organism or database specific configuration. To do that, create your class (perhaps in *bio/sources/chado-db/main/src/*) set the *processors* property in your source element. For example for reading the FlyBase Chado database there is a *FlyBaseProcessor* which can be configured like this:

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

Current uses FlyMine uses the *chado-db* source for reading the ''Drosophila" genomes from the FlyBase *chado* database. The *FlyBaseProcessor* sub-class is used for configuration and to handle FlyBase special cases.

modMine for the modENCODE project uses *ChadoDBSource* for reading ''D. melanogaster" from FlyBase and to read ''C. elegans" data from the WormBase *chado* database. The *WormBaseProcessor* sub-class is used for configuration when reading from WormBase.

Implementation notes for the chado-db source The *chado-db* source is implemented by the *ChadoDBConverter* class which runs the *ChadoProcessor's that have been configured in the 'project.xml*. The configuration looks like this:

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

ChadoDBConverter.process() will create an object for each *ChadoProcessor* in turn, then call *ChadoProcessor.process()*.

Chado sequence module table processing *ChadoSequenceProcessor* processes the sequence module from Chado. The *process()* method handles each table in turn by calling: *processFeatureTable()*, *processFeatureCVTermTable()* etc.

Each table processing method calls a result set method, eg. *processFeatureTable()* calls *getFeatureTableResultSet()* and then processes each row. The returned *ResultSet* may not always include all rows from the Chado table. For example the *getFeatures()* method returns a sub-set of the possible feature types and that list is used to when querying the feature table.

Generally each row is made into an appropriate object, eg. in *processFeatureTable()*, *feature* table rows correspond to *BioEntity* objects. Some rows of some tables are ignored (ie. not turned into objects) based on configuration.

Reading the feature table Handled by *ChadoSequenceProcessor.processFeatureTable()*

For each feature it calls: *ChadoSequenceProcessor.makeFeatureData()*, which may be overridden by subclasses. If *makeFeatureData()* returns null (eg. because the sub-class does not need that feature) the row is discarded, otherwise processing of the feature continues.

Based on the configuration, fields in the *BioEntity* are set using *feature.uniquename* and *feature.name* from Chado.

If the *residues* column in the feature is set, create a *Sequence* object and add it to the new *BioEntity*.

Reading the featureloc table Handled by *ChadoSequenceProcessor.processLocationTable()*.

This method gets passed a result set with start position, end position and information from the *featureloc* table. For each row from the result set it will:

- store a *Location* object
- set *chromosomeLocation* in the associated *SequenceFeature*
- set the *chromosome* reference in the *SequenceFeature* if the *srcfeature* from the *featureloc* table is a chromosome feature

Reading the feature_relationship table Handled by *ChadoSequenceProcessor.processRelationTable()*.

This method calls *getFeatureRelationshipResultSet()* to return the relations of interest. The relations will be used to create references and collections.

The method will automatically attempt to find and set the appropriate references and collections for *part_of* relations. As an example, if there is a *part_of* relation in the table between *Gene* and *Transcript* and there *Gene.transcript* reference or a *Gene.transcripts* collection, it will be set.

There are two modes of operation, controlled by the *subjectFirst* parameters. If true, order by the *subject_id* of the *feature_relationship* table so we get results like:

gene1_feature_id	relation_type	protein1_feature_id
gene1_feature_id	relation_type	protein2_feature_id
gene2_feature_id	relation_type	protein1_feature_id
gene2_feature_id	relation_type	protein2_feature_id

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

If *subjectFirst* is false we get results like:

gene1_feature_id	relation_type	protein1_feature_id
gene2_feature_id	relation_type	protein1_feature_id
gene1_feature_id	relation_type	protein2_feature_id
gene2_feature_id	relation_type	protein2_feature_id

The first case is used when we need to set a collection in the gene, the second if we need to set a collection in proteins.

Reading the cvterm table Handled by *ChadoSequenceProcessor.processFeatureCVTermTable()*

Using the default chado source

1. Add the chado database to your MINE_NAME.properties file, eg:

```
db.flybase.datasource.class=org.postgresql.ds.PGPoolingDataSource
db.flybase.datasource.dataSourceName=db.flybase
db.flybase.datasource.serverName=SERVER_NAME
db.flybase.datasource.databaseName=DATABASE_NAME
db.flybase.datasource.user=USER_NAME
db.flybase.datasource.password=SECRET_PASSWORD
db.flybase.datasource.maxConnections=10
db.flybase.driver=org.postgresql.Driver
db.flybase.platform=PostgreSQL
```

The chado database has to be on the local network.

2. Add source to project XML file

```
<source name="chado-db" type="chado-db">
  <property name="source.db.name" value="flybase"/>
  <property name="organisms" value="7227"/>
  <property name="dataSourceName" value="FlyBase"/>
  <property name="converter.class" value="org.intermine.bio.dataconversion.ChadoDBConverter"/>
  <property name="processors" value="org.intermine.bio.dataconversion.SequenceProcessor"/>
</source>
```

3. Run the build

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

See Database Building for more information on running builds.

This will load the data using the default chado loader. If you want to load more data you will have to write a custom chado converter. FlyMine uses a FlyBase chado "processor" to parse interactions, etc. See FlyBaseProcessor.java for an example.

Ensembl This page describes how to load Ensembl 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:

ftp://ftp.ensembl.org/pub/current_mysql

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
db.ensembl.9606.core.datasource.databaseName=homo_sapiens_core_59_37d
db.ensembl.9606.core.datasource.species=homo_sapiens
db.ensembl.9606.core.datasource.user=DB_USER
db.ensembl.9606.core.datasource.password=DB_PASSWORD

# variation database
db.ensembl.9606.variation.datasource.serverName=SERVER_NAME
db.ensembl.9606.variation.datasource.databaseName=homo_sapiens_variation_59_37d
db.ensembl.9606.variation.datasource.species=homo_sapiens
db.ensembl.9606.variation.datasource.user=DB_USER
db.ensembl.9606.variation.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*

```
$ ./ensembl.pl MINE_NAME TAXONID DATA_DESTINATION
```

for example:

```
$ ./ensembl.pl flymine 7165 /data/ensembl/current
```

Load XML file into InterMind database

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

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

When you run a database build, every XML file in the directory specificed 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

```
<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
src.data.dir	location of the fasta data file	these data will be loaded into the database
includes	name of data file	this data file will be loaded into the database
sequenceType	class name	type of sequence to be loaded
loaderClass-Name	name of Java file that will process the fasta files	only use if you have created a custom fasta loader

GFF3 InterMine comes with a GFF parser which loads GFF3 data files into your mine - without writing any Perl or Java code. This isn't a source itself but genome annotation from gff files can be loaded easily by creating a new source of type gff. See redfly, malaria-gff and tiffin for examples.

Configuration is added to the *project.properties* file and an optional handler can be added to deal with data in the attributes section of the gff file.

Types of data loaded sequence features

How to download the data N/A - will parse any file in GFF3 format

How to load the data into your mine

1. place valid GFF3 files into a directory
2. add entry to project XML file
3. run build

```
# example GFF3 file
MAL1    ApiDB   gene    183057  184457  .       -       .       ID=gene.46311;description=hypothetica
MAL1    ApiDB   mRNA    183057  184457  .       +       .       ID=mRNA.46312;Parent=gene.46311
```

If you follow the above steps with this data file, the following will happen:

1. gene and mRNA objects created
2. "MAL1" will be the identifier
3. start = 183057, end = 184457
4. gene will be located in -1 strand, mRNA will be located on the 1 strand.

To do more processing or access the attributes, eg. the "ID=mRNA.46312;Parent=gene.46311" string, you will have to write your own GFF3 parser. See below for details.

Project XML Here is an example GFF3 entry in the project XML file:

```
    # add to project.xml file
    # NOTE: update the "type" if you are using your own custom GFF3 parser

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

Here are the descriptions of the properties available:

property	example	definition
gff3.seqClsName	Chromosome	the ids in the first column represent Chromosome objects, e.g. MAL1
gff3.taxonId	36329	taxon id 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)

Writing a custom GFF parser You can extend the generic parser by writing your own Java code to process the GFF3 data.

Make Source script Create your custom source by running the create source script:

```
$ ./bio/scripts/make_source mouse-cdna gff
created /home/USER_NAME/git/bio/sources/mouse-cdna directory for mouse-cdna
```

The script has created a new source for you in the *bio/sources* directory.

Java code The Java file you now want to edit is here: *bio/sources/SOURCE_NAME/main/src/org/intermine/bio/dataconversion*

The *process()* method is called for every line of GFF3 file(s) being read. Features and their locations are already created but not stored so you can make changes here. Attributes are from the last column of the file are available in a map with the attribute name as the key. For example:

```
Item feature = getFeature();
String symbol = record.getAttributes().get("symbol");
feature.setAttribute("symbol", symbol);
```

Any new Items created can be stored by calling addItem(). For example:

```
String geneIdentifier = record.getAttributes().get("gene");
gene = createItem("Gene");
gene.setAttribute("primaryIdentifier", geneIdentifier);
addItem(gene);
```

You should make sure that new Items you create are unique, i.e. by storing in a map by some identifier.

It may be helpful to look at current GFF3 parsers:

1. *LongOligoGFF3RecordHandler.java*
2. *MirandaGFF3RecordHandler.java*
3. *RedFlyGFF3RecordHandler.java*
4. *FlyRegGFF3RecordHandler.java*
5. *DrosDelGFF3RecordHandler.java*

See Tutorial for more information on how to run a GFF source.

Identifier Data Sources You can load MODs ids into your mine using identifier data sources.

Types of data loaded genes

How to download the data

flybase-identifiers http://flybase.org/static_pages/downloads/FB20XX_XX/synonyms/fb_synonym_fb_FB20XX_XX.tsv.gz - where FB20XX_XX = the current FlyBase release

zfin-identifiers http://zfin.org/downloads/ensembl_1_to_1.txt

sgd-identifiers http://downloads.yeastgenome.org/curation/chromosomal_feature/SGD_features.tab

wormbase-identifiers query wormbase biomart webservice

mgi-identifiers ftp://ftp.informatics.jax.org/pub/reports/MGI_Coordinate.rpt

rgd-identifiers ftp://rgd.mcw.edu/pub/data_release/GENES_RAT.txt

How to load the data into your mine project XML example

```
 <source name="flybase-identifiers" type="flybase-identifiers">
   <property name="src.data.dir" location="/DATA/flybase-identifiers"/>
 </source>

 <source name="zfin-identifiers" type="zfin-identifiers">
   <property name="src.data.dir" location="/DATA/zfin-identifiers"/>
 </source>

 <source name="sgd-identifiers" type="sgd-identifiers">
   <property name="src.data.dir" location="/DATA/sgd-identifiers"/>
 </source>

 <source name="wormbase-identifiers" type="wormbase-identifiers">
   <property name="src.data.dir" location="/DATA/worm-identifiers"/>
 </source>

<source name="mgi-identifiers" type="mgi-identifiers">
   <property name="src.data.dir" location="/DATA/mgi-identifiers"/>
 </source>

<source name="rgd-identifiers" type="rgd-identifiers">
   <property name="src.data.dir" location="/DATA/rgd-identifiers"/>
 </source>
```

InterMine items XML Use this source to load InterMine Items XML Format conforming to the data model directly into the production database.

intermine-items-xml-file Use this source to load InterMine Items XML Format conforming to the data model directly into the production database.

intermine-items-large-xml-file Use this source to load InterMine Items XML Format 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 Use the Perl Items API to generate an InterMine Items XML Format file. Add an entry to your project XML file pointing to this new data file.

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

```
<source name="entrez-organism" type="entrez-organism">
  <property name="src.data.file" location="build/organisms.xml"/>
</source>
```

SNP Load SNP data from a downloaded mysql database

Types of data loaded SNPs, chromosomes

How to download the data Genetic and protein interaction data from BioGRID

How to load the data into your mine

```
<source name="ensembl-snp-db-human" type="ensembl-snp-db" dump="true">
  <property name="source.db.name" value="ensembl.9606.variation" />
  <property name="organism" value="9606" />
  <property name="sources" value="dbSNP" />
</source>
```

Sequence Ontology (SO) This source loads no data but adds a class in the data model for every term in the sequence ontology in your data model. SO terms represent biological features such as gene, exon, 3' UTR. You should include this source if you are loading genome annotation.

Types of data loaded Sequence Ontology terms

How to download the data Included in InterMine source code

How to load the data into your mine project XML example

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

```
<source name="uberon" type="uberon">
  <property name="src.data.file" location="/data/uberon/uberon.obo"/>
</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/flymine*.

- 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
- rnai
- 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, specifically to import data from an InterMine data format XML file. This is achieved in two parts; the first creates the files to describe the new source while the second configures an individual mine to use these files. The XML file containing the data could be created in any language. InterMine includes Java and Perl APIs to assist.

Thanks to Anthony Smith from the MRC Dunn Human Nutrition Unit for providing this tutorial.

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 Java Items API Java Items API to create and store items to the database.

intermine-items-xml-file This type of source can read a in InterMine XML format and store the results in a mine. It should be configured like this in the *project.xml* file for the mine:

```
<source name="my-new-source-name" type="">
  <property name="src.data.file" location="/some/directory/objects_in_intermine_format.xml"/>
</source>
```

The InterMine Perl Items API can be used to generate these XML files. Or you can create XML from any language. 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=org.postgresql.jdbc3.Jdbc3PoolingDataSource
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.

```
// loop through each row of results
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*.

The db value has to match the ''source.db.name'' in your project XML entry, for example:

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

```
# flymine.properties

db.flybase.datasource.class=org.postgresql.ds.PGPoolingDataSource
db.flybase.datasource.dataSourceName=db.flybase
db.flybase.datasource.serverName=LOCALHOST
db.flybase.datasource.databaseName=FB2011_01
db.flybase.datasource.user=USERNAME
db.flybase.datasource.password=SECRET
db.flybase.datasource.maxConnections=10
db.flybase.driver=org.postgresql.Driver
db.flybase.platform=PostgreSQL
```

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 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 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:

```
<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):

```
<class name="Protein" is-interface="true">
  <collection name="features"  referenced-type="NewFeature" reverse-reference="protein"/>
</class>
```

The final additions XML should look like:

```
<?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.

Keys 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:

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

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

```
<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.

The data file should have the same format as the XML below:

```
<items>
   <item id="0_1" class="" implements="http://www.intermine.org/model/bio#NewFeature>
      <attribute name="identifier" value="feature2" />
      <attribute name="confidence" value="0.8" />
      <reference name="protein" ref_id="0_3" />
  </item>
  <item id="0_2" class="" implements="http://www.intermine.org/model/bio#NewFeature>
      <attribute name="identifier" value="feature2" />
      <attribute name="confidence" value="0.37" />
      <reference name="protein" ref_id="0_3" />
```

```
  </item>
  <item id="0_3" class="" implements="http://www.intermine.org/model/bio#Protein">
      <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>
```

An example of a perl script to create InterMine data XML can be found at: *bio/scripts/intermine_items_example.pl*

If you are using data from a database:

```
<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.

Items APIs

Contents

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.

Usage in a Converter Items are most commonly used in a converter which provides some convenience methods.

Items are usually manipulated in a converter as part of a source InterMine source. All converters subclass *DataConverter* or one of its subclasses. This provides some convenience methods.

Create an item - this uses an *ItemFactory* (see below) which validates the class name and all fields against the data model:

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

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

```
store(gene);
store(items);
```

For a simple example of a converter see the *wormbase-identifiers* converter.

Item methods Item has methods to set values of attributes, references to other objects and collections of other objects.

To set an attribute (a field of an Item that is a Java type, e.g. String, Integer) use *setAttribute*). Note that all attribute types are treated as a String, they will be parsed to the appropriate type later.

```
gene.setAttribute("symbol", "zen");
organism.setAttribute("taxonId", "7227");
```

All items have an identifier generated for them automatically, these are used to reference other Items. You can set a reference with to an Item identifier or by using the item itself.

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

Or:

```
gene.setReference("organism", organism);
```

Set collections of other Items:

```
List<Item> publications = new ArrayList<Item>();
publications.add(pub1);
publications.add(pub2);
gene.setCollection(publications);
```

Or add one at a time:

```
gene.addToCollection("publications", pub1);
gene.addToCollection("publications", pub2.getIdentifier());
```

Attribute, Reference and ReferenceList (collections) can all be created independently and added to Items, this is sometimes useful in parsers to avoid holding too many Items in memory.

Creating Items with an ItemFactory When not used in a Converter you should create Items using an ItemFactory (the Converter does this for you), this validates the class name and all attribute/reference names against the data model. This requires that you get a Model instance (if there isn't already one).

```
Model model = Model.getInstance("genomic");
ItemFactory factory = new ItemFactory(model);
```

Create an item with the class name.

```
Item gene = itemFactory.makeItemForClass("Gene");
Item organism = itemFactory.makeItemForClass("Organism");
```

Reading/Writing XML To write a collection of Items to XML use *FullRenderer*:

```
FileWriter fw = new FileWriter(new File(fileName));
fw.write(FullRenderer.render(items));
fw.close();
```

To read an XML file into a List of items use *FullParser*:

```
List items = FullRenderer.parse(new FileInputStream(file));
```

Perl Items API

In the *intermine/perl* directory we provide a Perl library for creating files in InterMine "Item XML" format. Files in this format can be loaded into an InterMine database by creating a "source".

Usage Most code using these modules will follow this pattern:

Make a model

```
my $model = InterMine::Model->new(file => $model_file);
```

Make a new InterMine item XML document:

```
my $document = InterMine::Item::Document->new(
  model  => $model,
  output => $out_file,
);
```

Make an item:

```
my $gene = $factory->make_item("Gene");
```

Set some attributes

```
$gene->set(identifier => "CG10811");
```

or references:

```
my $org = $factory->make_item("Organism");
$org->set(taxonId => 7227);
$gene->set(organism => $org);
```

or collections:

```
$gene->set(transcripts => [$transcript1, $transcript2]);
```

It is also possible to combine creation and attribute setting in one command:

```
my $gene = $factory->make_item(
  'Gene',
  identifier  => 'CG10811',
  organism    => $org,
  transcripts => [$transcript1, $transcript2],
);
```

Repeat 4 as necessary then call *$document->write* to write the items to the output.

FlyMine example Example using the FlyMine model:

```
use InterMine::Model;
use InterMine::Item::Document;

my $model_file = $ARGV[0] or die;

my $model    = InterMine::Model->new(file => $model_file);
my $document = InterMine::Item::Document->new(model => $model);

my $organism = $document->add_item(
    'Organism',
```

```
    taxonId => 7227,
);

my $pub1 = $document->add_item(
    'Publication',
    pubMedId => 11700288,
);
my $pub2 = $document->add_item(
    'Publication',
    pubMedId => 16496002,
);

my $gene = $document->add_item(
    'Gene',
    identifier   => "CG10811",
    organism     => $organism,
    publications => [$pub1, $pub2]
);

# write as InterMine Items XML
$document->write();
```

Output:

```
<items>
   <item id="0_4" class="" implements="Gene">
      <attribute name="identifier" value="CG10811" />
      <collection name="publications">
         <reference ref_id="0_2" />
         <reference ref_id="0_3" />
      </collection>
      <reference name="organism" ref_id="0_1" />
   </item>
   <item id="0_1" class="" implements="Organism">
      <attribute name="taxonId" value="7227" />
   </item>
   <item id="0_2" class="" implements="Publication">
      <attribute name="pubMedId" value="11700288" />
   </item>
   <item id="0_3" class="" implements="Publication">
      <attribute name="pubMedId" value="16496002" />
   </item>
</items>
```

Example In *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

InterMine Items XML Format

The InterMine system has a file format for storing objects, which can be used to transfer data from one place to another. This document describes the file format.

The Data File Format The data is described in an XML file, with very few defined tags, and very little hierarchy. The document root tag is "<items>", which contains no attributes. The entire of the rest of the document is a list of "<item>" tags, each describing a single object to be stored in the file. The "<item>" tag has two attributes:

- *id* - this is a String, which must be unique to each individual item.
- *class* - this is the class name of the object.

The "<item>" tag has sub-elements corresponding to the values of the object's fields. These are "<attribute>", "<reference>", and "<collection>".

The "<attribute>" tag This tag describes a value of an attribute in the object. The tag has two attributes:

- *name* - this is the name of the attribute in the object.
- *value* - this is the value of the attribute in the object.

The "<reference>" tag This tag describes a value of a reference in the object. The tag has two attributes:

- *name* - this is the name of the reference in the object.
- *ref_id* - this contains the ID of the object that is referenced by this reference. The ID is the value that is set in the "<item>" tag's "id" attribute for the object being referenced.

The "<collection>" tag This tag describes a value of a collection in the object, which contains references to multiple objects. The tag has only one attribute:

- *name* - this is the name of the collection in the object.

The "<collection>" tag has sub-elements which are "<reference>" tags. These tags are identical to the "<reference>" tags defined above, except they do not have a "name" attribute - that is already defined by the "<collection>" tag.

A short example

```
<?xml version="1.0"?>
<items>
        <item id="1" class="Person">
        <attribute name="name" value="Fred"/>
        <reference name="address" ref_id="2"/>
        </item>
        <item id="2" class="Address">
        <attribute name="address" value="Somewhere"/>
        </item>
</items>
```

A fuller example file is our RedFly testing objects file at *bio/sources/flymine/redfly/test/resources/RedFlyGFF3RecordHandlerTest.xml*.

Generating InterMine XML With have a set of Perl modules to assist in the creation of InterMine XML.

Id Resolvers

ID resolvers hold data about identifiers, synonyms and cross referencess for a particular class (e.g. gene). The ID resolver provides methods to resolve any identifier to the unique primary identifier.

ID resolvers available in InterMine:

EntrezGeneIdResolverFactory	NCBI gene info for a collection of organisms	ftp://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz
FlyBaseIdResolverFactory	flybase chado db, for ''D.melanogaster" only	ftp://ftp.flybase.net/releases/current/psql flybase chado
WormBaseChadoIdResolverFactory	wormbase chado db, for ''C.elegans" only	modENCODE specific
ZfinIdentifiersResolverFactory	fish ids	http://zfin.org/downloads/ensembl_1_to_1.txt - ZFIN Marker associations to Ensembl IDs
MgiIdentifiersResolverFactory	mouse ids	ftp://ftp.informatics.jax.org/pub/reports/MGI_Coordinate.rpt
RgdIdentifiersResolverFactory	rat ids	ftp://rgd.mcw.edu/pub/data_release/GENES_RAT.txt
HgncIdResolverFactory	HGNC human gene ids	http://www.genenames.org/cgi-bin/hgnc_downloads.cgi
EnsemblIdResolverFactory	Ensembl id	customised

Using ID Resolvers in InterMine data converters

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

1. Download the identifier file - ftp://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz
2. Unzip the file to */DATA_DIR/ncbi/gene_info*
3. Create a sub directory */DATA_DIR/idresolver/* as file root path and a symbolic link *entrez* to the file

```
$ cd /DATA_DIR/idresolver/
$ ln -s /DATA_DIR/ncbi/gene_info entrez
```

4. Add the root path to the file in *~/.intermine/MINE.properties*

```
resolver.file.rootpath=/DATA_DIR/idresolver/
```

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

number of matches	returns
0	NULL
1	new identifier
>1	NULL

Using ID Resolvers in your data converters

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

```
resolver.file.rootpath=/DATA_DIR/idresolver/
```

the key and the symbolic link of the data file need to be hard-coded in factory class, e.g. in *EntrezGeneIdResolverFactory*

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

As for database case, e.g. flybase chado

```
# chado DB for flybase data

db.flybase.datasource.class=org.postgresql.jdbc3.Jdbc3PoolingDataSource
db.flybase.datasource.dataSourceName=db.flybase
db.flybase.datasource.serverName=NAME
db.flybase.datasource.databaseName=DBNAME
db.flybase.datasource.user=USER
db.flybase.datasource.password=PWD
db.flybase.datasource.maxConnections=10
db.flybase.driver=org.postgresql.Driver
db.flybase.platform=PostgreSQL
```

the key also needs to be hard-coded in factory class, e.g. in FlyBaseIdResolverFactory

```
private final String propName = "db.flybase";
```

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, in the meanwhile, it also reads and parses data from a file or db containing id information, and last saves them to a java map in IdResolver.

The new factory class need to inherit super class IdResolverFactory. To implement reading and parsing data from a customized file/db, please refer to the existing factories.

Future Plans

- data file path will be simplified
- generalized resolver factory which will read a configuration file to be aware identifier information by column. e.g. type=tab, column.0=mainId, etc.
- more efficient and smarter caching

1.5.3 Database Building

A 'build' of a mine is a complete data loading run starting from an empty database. It is recommended that you use the project_build script. The build script runs the data integration and any post-processing steps.

Each mine has an integrate project that reads the project.xml file and builds the data warehouse. This steps through each *source* defined in the project.xml file and transates the specified data from a source format and loads it into the production database. Data integration is governed by primary keys, any conflicts are resolved by a priorities config file.

project_build script

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:

```
<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

project_build allow the following options:

-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) [1].
-t	instead of dumping and reloading, make backup copies of the database in the server using the "CREATE DATABASE foo WITH TEMPLATE bar" command
-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*

Data Integration

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
```

Integration can be run from the *MINE_NAME/integrate/* directory by:

```
$ ant -v -Dsource=all
```

To run just one *source* specify the name (as it appears in project.xml):

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

[1] this allows multiple sets of properties file. eg. in the malariamine case, passing ''-V 1.0'' causes the build system to look for *build.properties.malariamine.1.0* and *malariamine.properties.1.0* instead of the default files

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

```
$ ant -v -Dsource=malaria-gff -Daction=[retreive|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.

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:

```
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*:

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

From the uniprot source (*uniprot_additions.xml*):

```
...
<class name="Protein" is-interface="true">
  <attribute name="description" type="java.lang.String"/>
  <attribute name="ecNumber" type="java.lang.String"/>
  <collection name="publications" referenced-type="Publication"/>
</class>
...
```

Final, merged, model definition:

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

The resulting class has all attributes of the *Protein* from *core.xml* and from *uniprot_additions.xml*. Note that in uniprot we don't need to declare a base class for *Protein* (like as *extends="BioEntity"*) as the base class from *core.xml* is merged into the final class.

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 this cases there should usually be a key defined for the referenced class as well.

Here is an example from the *uniprot* source: [source:trunk/bio/sources/uniprot/resources/uniprot_keys.properties uniprot_keys.properties]

It is still possible to use a legacy method of configuring keys, where keys are defined centrally in *dbmodel/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
Gene.key_ncbiGeneNumber=ncbiGeneNumber
Protein.key_identifier_org=identifier, organism
Protein.key_primaryacc=primaryAccession
```

Using global keys in each source In this case the properties file for each data source lists primary key names from the 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.

Data integration

During data integration, the objects created by each source are added to the database. If no primary keys are set all objects are added. However, if a source has a primary key defined then the build system will check the database for that primary key. If the key is not found, the object is added. If the key is present in the database, the build system will attempt to merge the new object from the current source with the existing object in the database.

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:

```
$ ant -v
```

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

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

When running one postprocess step like this, 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. We are trying to eliminate the need to use this.

Should I use it? Yes, for the moment if you are using standard InterMine sources.

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.

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-bioseg-location-index Create a bioseg 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 and have [BiosegInstallation bioseg] installed.

create-overlap-view Replace the *sequencefeatureoverlappingfeatures* table with a view that uses the [BiosegInstallation bioseg] type to calculate the overlaps.

Should I use it? Yes, if you have genome annotation and would like to query overlaps and have [BiosegInstallation bioseg] installed.

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-search-index Creates the lucene search index used by the webapp.

Should I use it? Yes, if you are releasing a webapp.

create-attribute-indexes Create indexes on all attributes to help speed up queries.

Should I use it? Always. It should be run last of all post-processing steps.

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 ObjectStoreSummaryProperties for more information.

Should I use it? - Always.

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.

1.5.4 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@blackhol
```

Comparing against two mines is as above, except you simply need to add a second service location:

```
python compare_templates_for_releases.py www.flymine.org/flymine beta.flymine.org/beta you@your.host
```

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%
BDGP_Gene                                       release-beta:     260, release-28.0:      62, diff
ChromLocation_CRMOverlappingTFBindingsite       release-beta:      42, release-28.0:     213, diff

------------------------------------------------------------------------
-----------------------------------Only in 28.0:
ChromosomeLocation_Tiffin                          8
Disease_GeneOrthologue                           363
ESTclone_LocationOverlappingGeneOrthologue_new    93
ESTclone_LocationOverlappingGeneStructure          4
Gene_Inparalogue                                  11
Gene_Tiffin                                      156
Probe_Gene                                         1
TiffinBSmotif_expressionTerm                      49
TiffinBSmotif_genes                             1356
TiffinBSmotif_locations                           23
------------------------------------------------------------------------
-----------------------------------Only in beta:
Amplicon_RNAiResults                              39
Gene_AdjacentGene_FlyAtlas_downstream              0
Gene_OverlapppingGenes                             1
Genes_Publications                            126002
Organism_interologs                              278
-------------------------------------------------In Both: Diff < 10%
All_Genes_In_Organism_To_Publications           release-beta:  126002, release-28.0:  121503, diff
AlleleClass_Allele                              release-beta:    2132, release-28.0:    2117, diff
```

1. ‘‘‘In Both: Diff >= 10%’‘‘ - templates run in both mines and result counts returned were very different.

2. ‘‘‘Only in’‘‘ - template was found in one mine and not the other.
3. ‘‘‘In Both: Diff < 10%’‘‘ - template run in both mines and results returned were different. It’s probably safe to assume these are okay.

Acceptance Tests

How to run the tests

1. Add a file to *MINE_NAME/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:

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

Or doesn’t have any results:

```
no-results {
        sql: select * from datasource where url is null or name is null or description is null
        note: all fields of data source should be filled in
}
```

Or has at least some results:

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

1.5.5 Data loading performance

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

Storing Items in order

When loading objects into the production ObjectStore the order of loading can have a big impact on performance. It is important to store objects before any other objects that reference them. For example, if we have a Gene with a Publication in its evidence collection and a Synonym referencing the Gene, the objects should be stored in the order: Publication, Gene, Synonym. (If e.g. the Gene is stored after the Synonym a placeholder object is stored in the Gene’s place which is later replaced by the real Gene. This takes time).

Objects are loaded in the order that Items are stored by converter code or the order they appear in an Items XML file. When Items are stored into the items database (during the build or using *ant -Dsource=sourcename -Daction=retrieve*) you can check if there are improvements possible with this SQL query:

```
SELECT classnamea, name, classnameb, count(*)
FROM (SELECT distinct itema.classname AS classnamea, name, itemb.classname AS classnameb, itemb.ident
      FROM item AS itemA, reference, item AS itemB
      WHERE itema.id = itemid AND refid = itemb.identifier
            AND itema.id < itemb.id) AS a
GROUP BY classnamea, name, classnameb;
```

If there are no results then no improvement can be made. The example below shows that there were 27836 Gene Items stored after the Synonyms that reference them. *subject* is the name of the reference in Synonym. Changing the store order would improve performance.

```
classnamea |  name   | classnameb | count
-----------+---------+------------+-------
Synonym    | subject | Gene       | 27836
```

Recommended Hardware and Software

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

- Install lots of RAM, like 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. FlyMine 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.
- Use a recent version of PostgreSQL. At the time of writing, Postgres 8.4 is promising to come out Real Soon Now with significant improvements for the type of query we run frequently, especially if the machine has a large RAID array. Make sure you check out the effective_concurrency option.
- We 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.

PostgreSQL Configuration

Recommended settings for PostgreSQL are in Installing PostgreSQL

1.6 Web Application

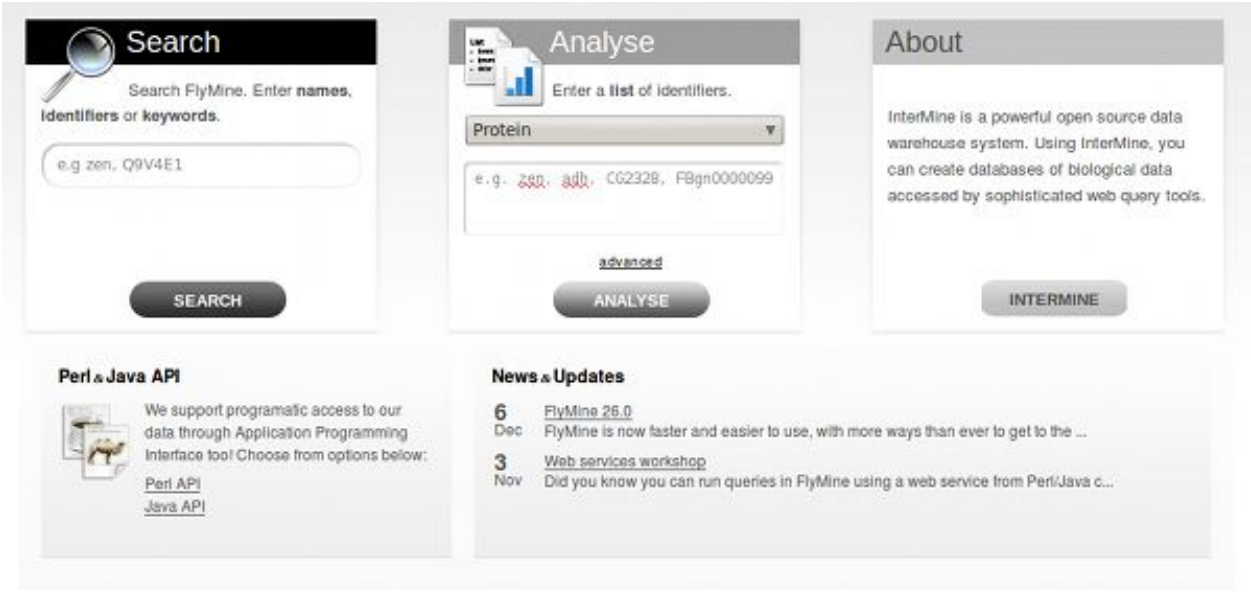
1.6.1 Home page

Note: This text describes how to customize the homepage of your mine.

See also:

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 customize the text these contain and how they are called. For example, we might want to change the title of the list upload box to say "List Upload" instead of the default "Analyse". We might also want to change the informative text the box contains and provide a set of example identifiers that one can analyse. Then, we would add the following text to the config file:

```
begin.listBox.title = List Upload
begin.listBox.description = Enter a <strong>list</strong> of identifiers.
begin.listBox.example = e.g. zen, adh, CG2328, FBgn0000099
```

Note: You will find that only the description field accepts HTML.

The first search box is configured in much the same way as the list upload one is:

```
begin.searchBox.title = Search
begin.searchBox.description = Search FlyMine. Enter <strong>names</strong>, <strong>identifiers</str
or <strong>keywords</strong> for genes, proteins, pathways, ontology terms, authors, etc. (e.g. \
<em>eve</em>, HIPPO_DROME, glycolysis, <em>hb</em> allele).
```

Basically, we have only changed listBox to say searchBox in the configuration.

The third/info box can contain a descriptive text about your mine or it can offer a link to a tour of the project. Take the example from FlyMine project:

```
begin.thirdBox.title = First Time Here?
begin.thirdBox.description = FlyMine integrates many types of data for <em>Drosophila</em>, \
<em>Anopheles</em> and other organisms. You can run flexible queries, export results and analyse list
data.
begin.thirdBox.link = http://www.flymine.org/help/tour/start.html
begin.thirdBox.linkTitle = Take a tour
```

By providing the .link parameter a button will be shown at the bottom of the box with a custom link of choice.

Since InterMine 0.96, you can serve up a custom text in the third "information" box to the user, based on whether they have visited the homepage before or not. We do this through a cookie that will, for a year, indicate for your computer, that the homepage has been visited.

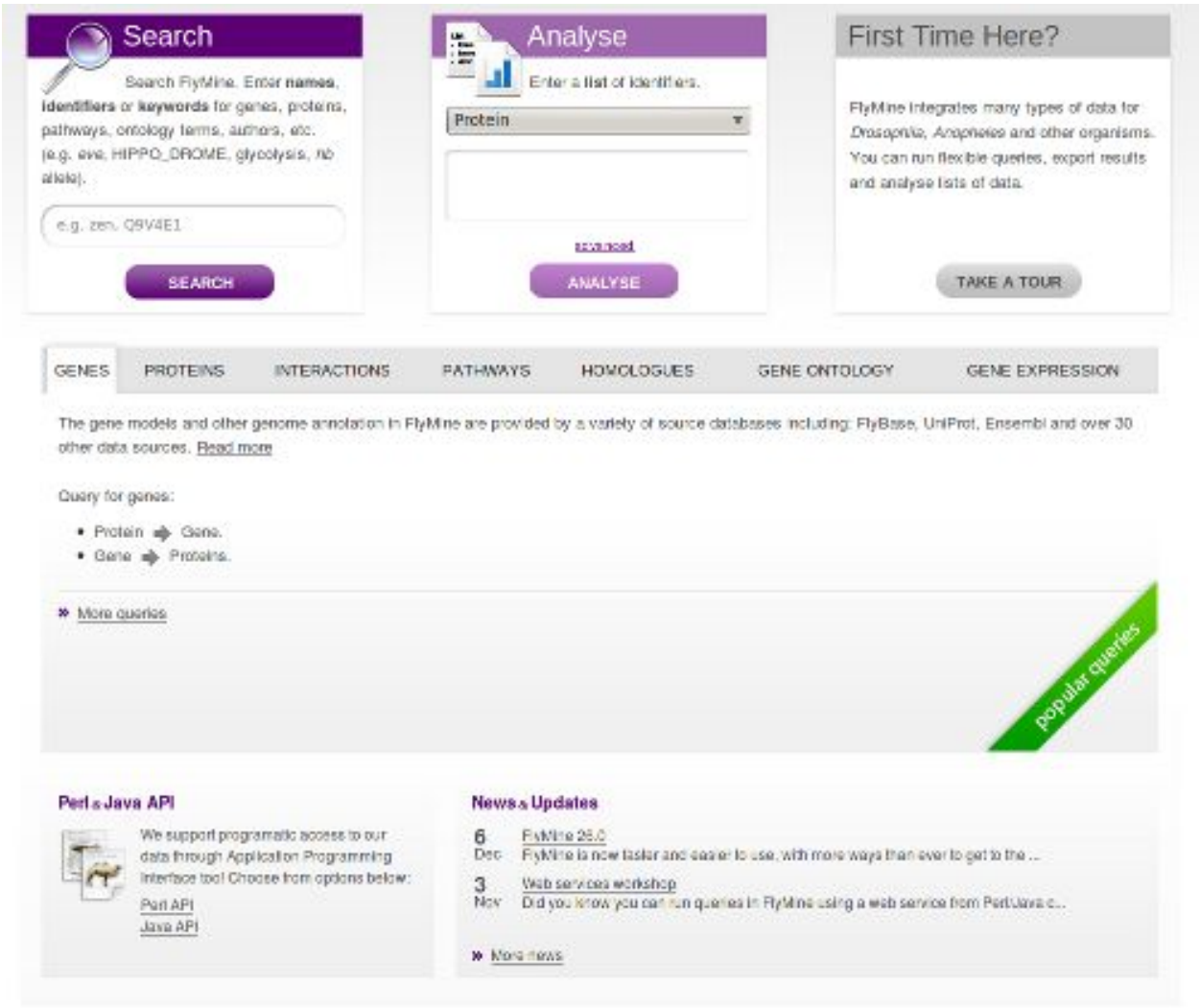
In order to change the values of the third box based on whether the user has visited the page or not, prepend the text "visited" before an uppercased key. For example, if one wanted to say "Welcome Back" instead of "First Time Here?" as the title of the box, we would add the following key=value pair:

```
begin.thirdBox.visitedTitle = Welcome Back
```

The fields that you do NOT set in this way, will simply show the text configured in the normal way. So even though someone has visited the homepage before, unless I add a "visited" property, the text stays the same as before.

Popular Templates Customization

By visiting the Templates page one can see a listing of templates in different categories. If you would like to show 10 most popular templates per category on your homepage, follow this text.



Configuring the popular templates is very straightforward. Let us say that we have a Genomics category of templates. To add it to the homepage, we will add the following to the config file:

```
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. The number in the config key specifies the order in which we want to show them. So if we have two categories, Genomics and Proteins, and they should appear in this order, we would write this:

```
begin.tabs.1.id = Genomics
begin.tabs.2.id = Proteins
```

The other customisation we can do is specify an informative text that is to appear in the tab above the templates listing (again, this text accepts HTML.):

```
begin.tabs.1.id = Genomics
begin.tabs.1.description = This is some descriptive text
```

The last thing we will show is how to specify a custom category name to show as a link on the tab (entirely optional):

```
begin.tabs.1.id = Genomics
begin.tabs.1.description = This is some descriptive text
```

```
begin.tabs.1.name = Genes
```

Featured Lists

Lists with tag `im:homepage` will be shown on the homepage below the templates listing in a natural order, and/or an order specified by `im:order:n`.

To change the description text associated with this set of lists, edit the properties file like so:

```
begin.listsBox.description = These are the best lists ever
```

RSS/Blog Feed

To add the RSS feed at the bottom right corner of the page, add the following to your MINE properties file (in `.intermine` file):

```
project.rss = http://<your_blog>/<your_feed_url>
```

eg:

```
project.rss=http://blog.flymine.org/?feed=rss2
```

Two latest entries will be shown in the box. If you want to provide a link underneath the entry listing to your blog, add the following to the config file:

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

1.6.2 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.

More

The faceted search system was implemented by Nils Kölling, a summer intern with InterMine. See the talk he gave (at http://www.flymine.org/download/talks/keyword_search_talk.pdf) for more technical details.

1.6.3 Lists

"class_keys.properties" 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. The page aims to describe how it works.

Users can paste any identifier into the list upload form; it can be an identifier, symbol or name. The list upload form runs a series of queries to try to match the identifier to an object in the database.

This is also how the LOOKUP constraint works.

The list upload form runs the three queries listed below until it finds a match for the user's identifiers. It's now possible to run all three queries every time regardless of if a match was returned. You may want to configure your mine to do this if your database contains lots of identifiers that are assigned to different objects, this option would allow your users to see more options - not just the first.

Queries

Default Query First, we query for the value in key fields. For example:

```
select * from gene
where name = 'adh' OR
  symbol = 'adh' or
primaryIdentifier = 'adh' or
secondaryIdentifier = 'adh';
```

If this query returned results, that object is added to our list and we are done.

If this query didn't return any results, we move on to the next step.

"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".

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

example list The example list is set in "bag.example.identifiers" property in WebProperties.

organisms All organisms in your database will be displayed here. You can set the default 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.

List analysis

fields displayed determined by webconfig-model.xml

export See Export

"Convert to a different type" Tag conversion template with *im:converter* tag.

"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 Template queries will appear on the appropriate type of list analysis page, eg. Gene –> Proteins template would appear on Gene list analysis pages.

widgets See: List Widgets

List Widgets

List Widgets Questions & Answers

Source files Source files for the List widgets client.

Using a temporary list on the fly

Requirements

1. InterMine Generic WebService Client Library from GitHub or InterMine CDN.
2. InterMine List Widgets Client Library from GitHub or InterMine CDN.
3. A mine that has the desired Enrichment Widget configured.
4. An API Access Key generated by logging in to MyMine and visiting the API Key tab, then clicking on Generate a new API key. This assumes that you do not want to automatically provide the API key as is the case of within mine embedding that can be seen for example here.

Code First require the JavaScript libraries needed to run the example. You probably have your own version of a Twitter Bootstrap compatible CSS style included on the page already.

```
<!-- dependencies -->
<script src="http://cdn.intermine.org/js/jquery/1.7/jquery.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/imjs.js"></script>
<script src="http://cdn.intermine.org/js/intermine/widgets/latest/intermine.widgets.js"></script>
```

The next step is defining a couple of variables.

```
var root = 'http://www.flymine.org/query';
var tokn = 'U1p3r9Jb95r2Efrbu1P1CdfvKeF'; // API token
var name = 'temp-list-from-js-query'; // temporary list name
```

Now we connect with the mine through InterMine JavaScript Library.

```
// Service connection.
var flymine = new intermine.Service({
    'root':  root,
    'token': tokn
});
```

Then we define the query whose results will be converted into a list later on.

```
// The query herself.
var query = {
    'select': [ 'symbol', 'primaryIdentifier' ],
    'from': 'Gene',
    'where': {
        'symbol': {
            'contains': 'ze'
        }
    },
    'limit': 10
};
```

Now we call the mine converting the results of the query into a list.

```
flymine.query(query, function(q) {
    // Save the query as a list.
    q.saveAsList({'name': name}, function(l) {
        // Now we have created a list under a name.
    });
});
```

Now, in the callback that has created the list, we can instantiate the List Widgets client and display the result.

```
var widgets = new intermine.widgets(root + '/service/', tokn);
// A new Chart List Widget for a particular list in the target #widget.
widgets.chart('flyfish', name, '#widget');
```

The only problem with this approach is that if we make this sort of call multiple times, we will fail on the second and subsequent ocassions as we will get a WebService exception telling us that the 'temporary' list name is taken. *Thus inspect the code of the example to see how to make a call to the service to delete the list if it exists*.

Defining custom actions on widget events In a mine context, List Widgets are configured automatically to e.g. display a Query Results when clicking on "Create a List".

Outside of a mine context, one needs to pass in what happens when one interacts with the Widgets. You can also decide whether to show/hide either/and/or title or description of the widget (for everything else use CSS).

Clicking on an individual match (Gene, Protein etc.) in popover window:

```
var options = {
    matchCb: function(id, type) {
        window.open(mineURL + "/portal.do?class=" + type + "&externalids=" + id);
    }
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Clicking on View results button in a popover window:

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

Clicking on Create list button in a popover window:

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

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

```
var options = {
    "title": false,
    "description": false
};
Widgets.enrichment('pathway_enrichment', 'myList', '#widget', options);
```

Showing a Results Table The example below assumes that you have resolved all Query Results dependencies and have a PathQuery in JSON/JavaScript format that you want to display in a `#container`:

```
// PathQuery needs to be in a JSON string format.
var pq = JSON.stringify(pq);
// use an instance of a Service or perhaps you already have one.
var service = new intermine.Service({'root': service, 'token': token});
// Create a new ResultsTable.
var view = new intermine.query.results.CompactView(service, pq);
// Say where to put it.
view.$el.appendTo("#container");
// Show it.
view.render();
```

List enrichment widgets statistics

Enrichment widgets are located on the list analysis page. There are a number of different types of enrichment widgets, but all list a term, a count and an associated p-value. The term can be something like a publication name or a GO term. The count is the number of times that term appears for objects in your list. The p-value is the probability that result occurs by chance, thus a lower p-value indicates greater enrichment.

Method The p-value is calculated using the Hypergeometric distribution. Four numbers are used to calculate each p-value:

```
     (M choose k) (N-M choose n-k)
P =   -----------------------------
                N choose n
```

n the number of objects in your list

N the number of objects in the reference population

k the number of objects annotated with this item in your list

M the number of objects annotated with item in the reference population

---

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.

Note: The relevant InterMine source.

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 per GO term CCGO is calculated as:

```
            LGO/LWH
CCGO = -----------------
            NGO/NWG
```

LGO Average gene length of genes associated with a GO term

LWG Average length of the genes in the whole genome

NGO Number of genes in the genome associated with this GO term

NWG Total number of genes in the whole genome.

Note: The relevant InterMine source.

References

GOstat: Find statistically overrepresented Gene Ontologies within a group of genes
Beissbarth T, Speed TP.
Bioinformatics. 6.2004; 20(9): 1464-1465.
PubMed id: 14962934

GO::TermFinder–open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes
Boyle EI, Weng S, Gollub J, Jin H, Botstein D, Cherry JM, Sherlock G.
Bioinformatics. 2004 Dec 12;20(18):3710-5. Epub 2004 Aug 5.
PubMed id: 15297299

Controlling the false discovery rate: a practical and powerful approach to multiple testing
Benjamini, Yoav; Hochberg, Yosef
Journal of the Royal Statistical Society. 1995, Series B (Methodological) 57 (1): 289–300.

Augmentation Procedures for Control of the Generalized Family-Wise Error Rate and Tail Probabilities for the Proportion of False Positives
van der Laan, Mark J.; Dudoit, Sandrine; and Pollard, Katherine S.
Statistical Applications in Genetics and Molecular Biology: Vol. 3 : Iss. 1, Article 15, 2004.

What's wrong with Bonferroni adjustments
Perneger, TV.
BMJ Publishing Group. 1998;316:1236.

Variable locus length in the human genome leads to ascertainment bias in functional inference for non-coding elements
Taher, L. and Ovcharenko, I. (2009), Bioinformatics. 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.

See also:

FlyMine list widgets example on Tinkerbin.

There are several list widgets (widgets from now on) available on the InterMine list analysis page, and they are configured in Data and Widget Configuration.

There are three categories of widgets:

table displays the counts from the list for the collection specified

graph displays a chart based on a dataset you specify

enrichment displays the p-values of objects that appear in your list

To add a widget to your mine:

1. add config to your `webconfig-model.xml` file
2. re-release your webapp
3. view widget in a list analysis page

Below are the details on how to configure each widget type.

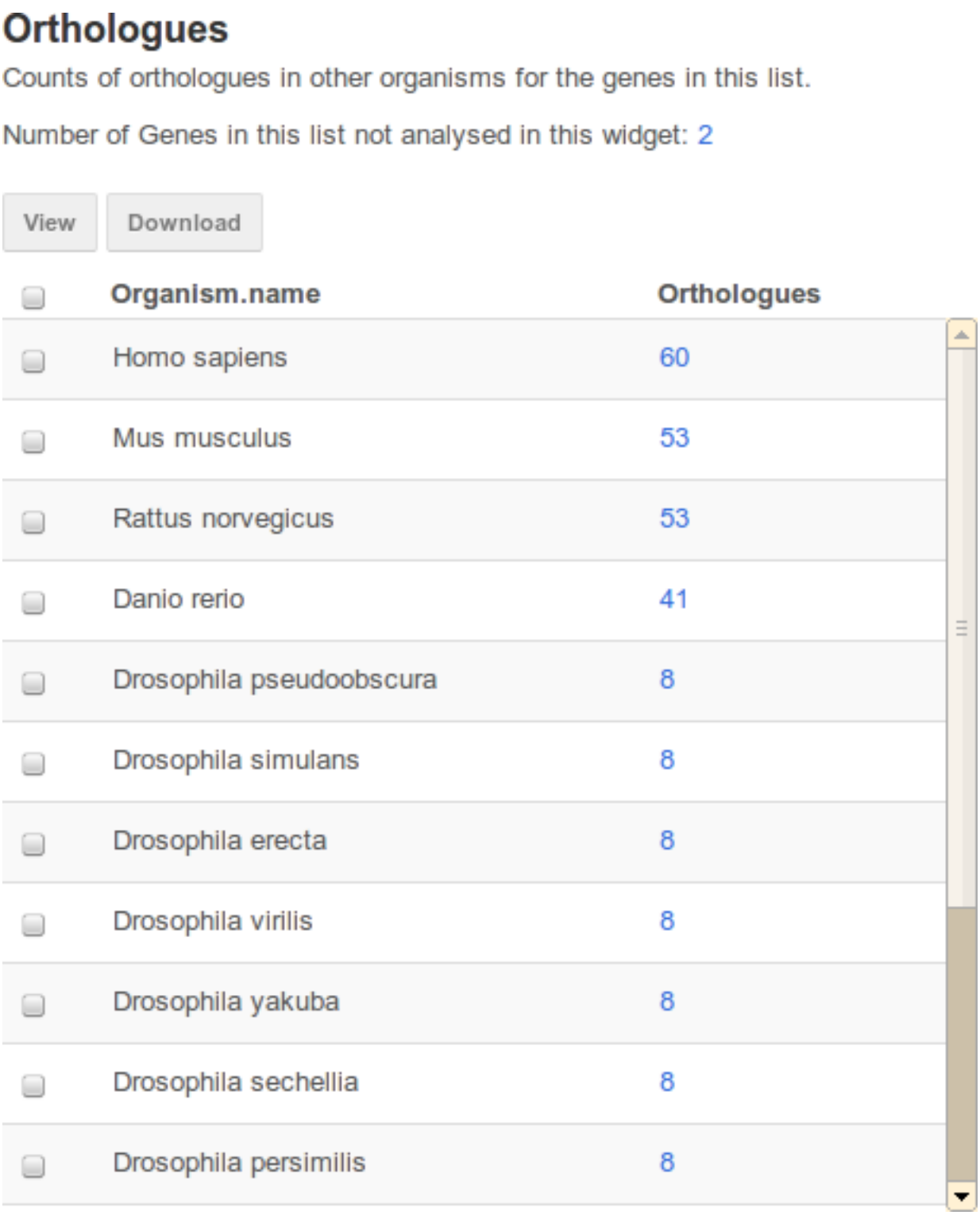
Note: Please read the documentation carefully and check your config file for typos. Most attributes are case sensitive. When the webapp is released, the config is validated and any errors displayed in the home page.

Configuration

Table widgets Table widgets display objects and the counts of related objects in your list.

An example table widget of Orthologues in FlyMine.

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



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

View Download

Organism.name	Orthologues
Homo sapiens	60
Mus musculus	53
Rattus norvegicus	53
Danio rerio	41
Drosophila pseudoobscura	8
Drosophila simulans	8
Drosophila erecta	8
Drosophila virilis	8
Drosophila yakuba	8
Drosophila sechellia	8
Drosophila persimilis	8

The following are optional attributes:

attribute	purpose	example
`title`	appears at the top of the widget	`Orthologues`
`description`	description of the widget	`Counts of orthologues`
`displayFields`	which fields from the objects in the collection (in the above example, `Gene.proteins`) to display, eg. `primaryAccession`	`name`
`columnTitle`	heading for the "count" column	`Orthologues`
`externalLink`	link displayed next to first column, identifier will be appended to link	
`externalLinkLabel`	label for external link	
`views`	path fields display in the query running when the user clicks on the widget	`symbol`

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

attribute	purpose	example
id	unique id used by javascript only. Spaces not allowed.	`unique_id`
graphType	which type of chart to render	`ColumnChart`,"BarChart" or `PieChart`
startClass	it's the root class for all the paths specified in the configuration [1].	`Gene`
typeClass	type of lists that should display this widget. Use the simple class name.	`Gene`
categoryPath	Must be attribute. We can specify the subclass using the syntax `path[subclass type]`	`mRNAExpressionResults.stageRange`
seriesPath	the series path. This has to be an attribute. We can specify the subclass using the syntax `path[subclass type]`	`mRNAExpressionResults.expressed`
seriesValues	the values of different series. Case sensitive. You can specify boolean values	`true`,`false` or `Up`,`Down`
seriesLabels	the labels displayed on the graphs to distinguish inside a category the different series	`Expressed`,`Not Expressed` or `Up`,`Down`
views	attributes paths displayed when the user clicks an area on the graph	`name`,`organism.name`

Warning: You can specify **only one** class in `typeClass`. If you need another type, you have to define a new widget.

The following are optional attributes:

[1] 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.

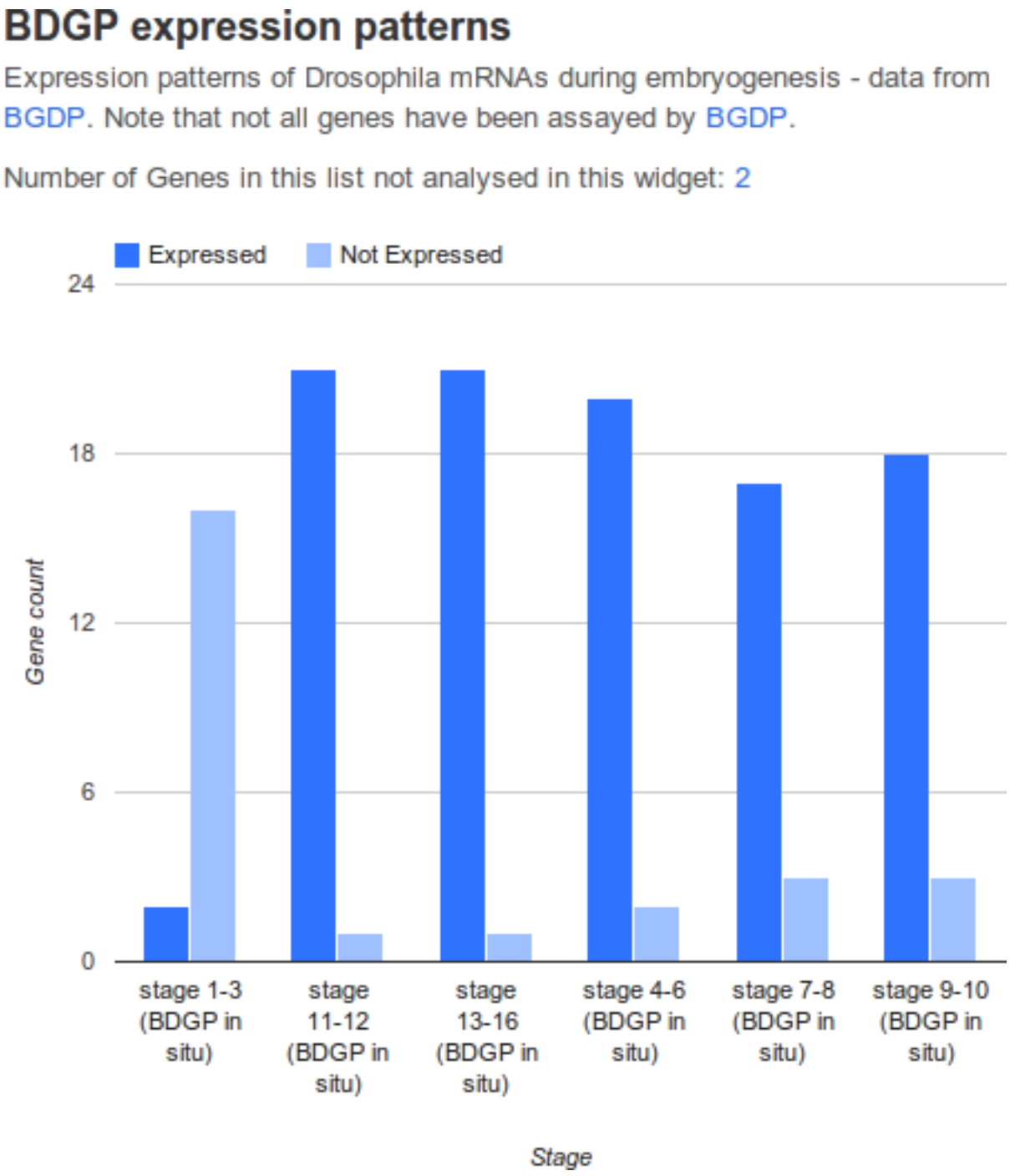


Fig. 1.1: An example chart widget of BDGP Expression Patterns in FlyMine.

attribute	purpose	example
`title`	appears at the top of the widget	`BDGP expression patterns`
`description`	description of the widget	`Expression patterns`
`domainLabel`	Label displayed on x-axis in the ColumnChart (on y-axis in the BarChart)	`Stage`
`rangeLabel`	Label displayed on y-axis in the ColumnChart (on x-axis in the a BarChart)	`Gene count`
`filterLabel`	label for filter form field	`Organism`
`filters`	the values for the filter, set in the dropdown [2].	`All,KEGG pathways,Reactome data`
`listPath`	the path used to build the bag constraint [3].	`FlyAtlasResult.material`
`constraints`	separated by comma, case sensitive, must be attributes, operator can be = or *!=* [4]	`organism.name=[Organism]` [5]

Note: The graphs use Google Visualitation API.

Enrichment widgets Enrichment widgets calculate p-values representing the probability annotation occurred by chance. See List enrichment widgets statistics for more information on how the p-value is calculated.

attribute	purpose	example
`id`	unique id used by JavaScript only. Spaces not allowed.	`unique_id`
`startClass`	Root class for all the paths specified in the configuration. Use simple name (e.g. Gene)	`Gene`
`startClassDisplay`	Field displayed when user clicks on the widget on 'Matches' column	`primaryIdentifier`
`typeClass`	Type of lists that should display this widget. Use the simple class name.	`Gene`
`enrich`	Field to be enriched, displayed in the widget in the firts column [6].	`goAnnotation.ontologyTerm.parents.name`
`views`	attributes paths displayed when the user clicks on *View results* button [6].	`symbol,organism.name`

Warning: You can specify **only one** class in `typeClass`. If you need another type, you have to define a new widget.

The following are optional attributes:

[2] 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.

[3] Optional if the `startClass` contains the bag type class.

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

[5] organism's name matching with the value selected in the filter with filterLabel 'Organism'

[6] You have to specify only one field. Specify the subclass using the syntax `path[subclass type]`.

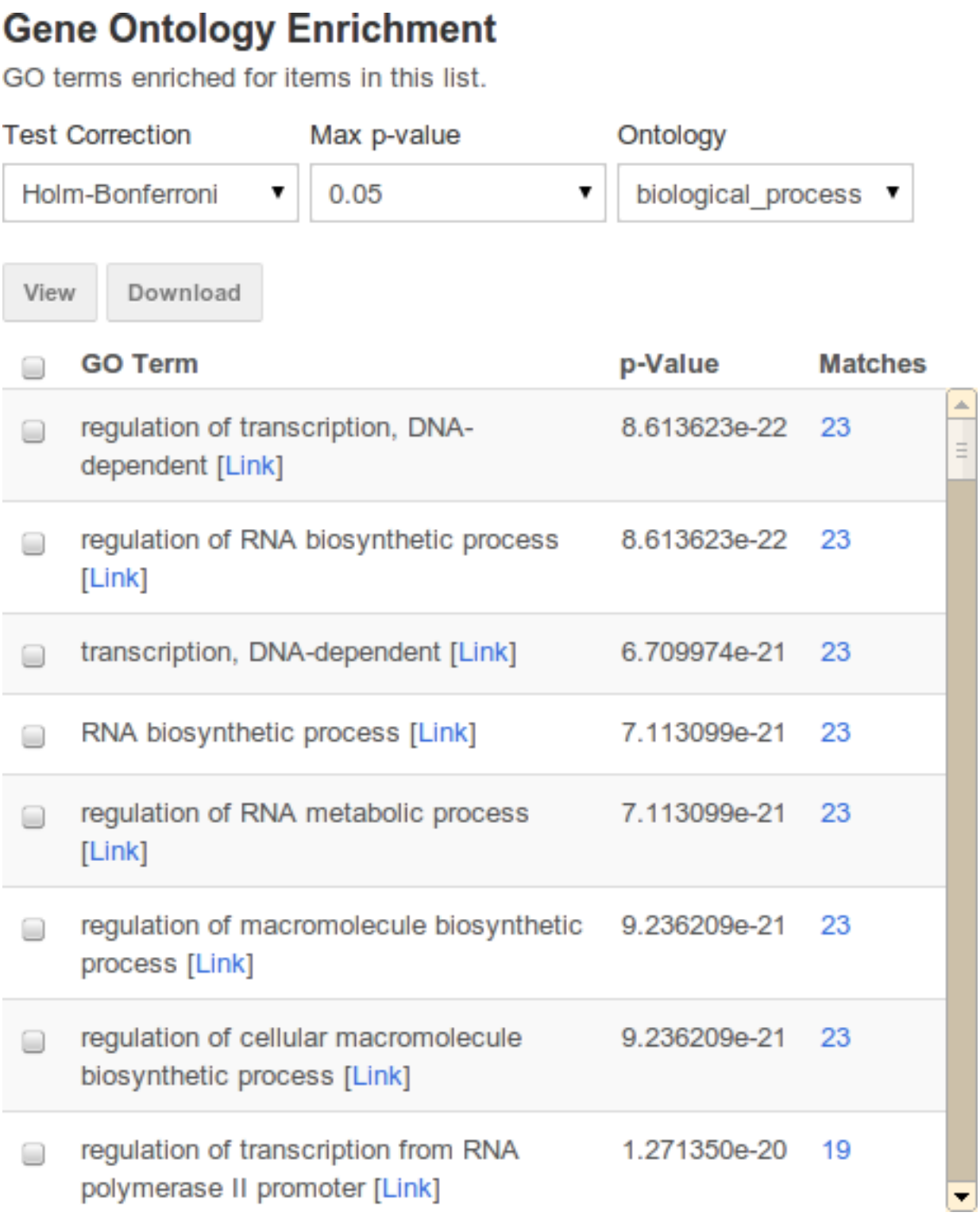


Fig. 1.2: An example enrichment widget of Gene Ontology in FlyMine.

attribute	purpose	example
`title`	appears at the top of the widget	`Gene Ontology Enrichment`
`description`	description of the widget	`GO terms enriched.`
`label`	heading for the column	`GO Term`
`externalLink`	link displayed next to first column	`googie`
`filters`	extra filters to add to the display [7]	`organism.name=[list]`
`filterLabel`	label for filter form field	`Ontology`
`enrichIdentifier`	identifier for the row displayed, if not specified, enrich field used [8].	`goAnnotation.ontologyTerm.identifier`
`constraints`	constraints separated by comma. The paths have to be attributes. The operator can be = or != [9].	`organism.name=[list]`
`constraintsForEnrichment`	constraints separated by comma used for building the query executed when the user clicks on the widget on 'Matches' column	`results.expressed = true`

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-mod` in metabolicMine

Displaying widgets

JavaScript

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

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

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

```
// Load all Widgets:
widgets.all('Gene', 'myList', '#all-widgets');
// Load a specific Chart Widget:
widgets.chart('flyfish', 'myList', '#widget-1');
// Load a specific Enrichment Widget:
widgets.enrichment('pathway_enrichment', 'myList', '#widget-2');
// Load a specific Table Widget:
widgets.table('interactions', 'myList', '#widget-3');
```

[7] Use static values or a grammar to specify the values contained in the list. The default value in general is the first value set in the 'filters' attribute or the first value returned by the query. With static values, you can add 'All' meaning no filter applied.

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

[9] Case sensitive. For the values we can use: static values the selected filter value using the syntax: `path contraint = [filter identifier]` only the value contained in the list.

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:

```
// Load all Widgets:
Widgets.all('Gene', 'myList', '#all-widgets');
// Load a specific Chart Widget:
Widgets.chart('flyfish', 'myList', '#widget-1');
// Load a specific Enrichment Widget:
Widgets.enrichment('pathway_enrichment', 'myList', '#widget-2');
// Load a specific Table Widget:
Widgets.table('interactions', 'myList', '#widget-3');
```

Where the *first parameter*‘ passed is either type of object or name of widget to load. The *second* is the name of list (public list) to access and *third* is an element on the page where your widgets will appear. This element needs to obviously exist on the page first. A common one is a div that would look like this: `<div id="all-widgets"></div>`.

7. Add HTML, eg:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>test</title>
    <script src="http://cdn.intermine.org/api"></script>
    <script type="text/javascript">
        intermine.load('widgets', function() {
```

```
                var Widgets = new intermine.widgets('http://beta.flymine.org/query/service/');
                // Load all Widgets:
                Widgets.all('Gene', 'myList', '#all-widgets');
            });
        </script>
    </head>

    <body>
        <!-- DIV goes here -->
        <div class="widget" id="all-widgets">
    </body>
    </html>
```

8. You will have noticed that the widgets either pickup a style (CSS) from your HTML page, or they appear unstyled. To style them, you can use a variant of Twitter Bootstrap.

1.6.4 Query Results

Content

Export

Default exporters in bio project

- tab/csv - tab or comma separated
- GFF3
- FASTA
- BED
- Galaxy - export data in tab or BED format to Galaxy server

Appears on these pages:

- Query results page
- List analysis page
- Genomic Region Search results page

Create a custom exporter

You can add additional export options to your InterMine by creating a custom exporter.

1. Write Java classes to handle the data

   see *BEDHttpExporter.java* and *BEDExporter.java*

2. Add the class to your webconfig-model.xml file:

```
<tableExportConfig id="bed" className="org.intermine.bio.web.export.BEDHttpExporter"/>
```

3. update Struts config

```
# MINE/webapp/resources/struts-config-model.xml
<action path="/bedExportAction" name="bedExportForm" type="org.intermine.web.struts.TableExportActio

# MINE/webapp/resources/tiles-defs-model.xml
<definition name="bedExportOptions.tile" path="/model/bedExportOptions.jsp" controllerUrl="/initSeque

# MINE/webapp/resources/struts-config-model-form.xml
<form-bean name="bedExportForm" type="org.intermine.bio.web.struts.BEDExportForm"/>
```

Customisation

By default, SequenceFeature and Protein can be exported as FASTA format

```
fasta.export.classes = SequenceFeature,Protein
```

Exporter description

```
exporter.galaxy.description = Export to Galaxy
exporter.sequence.description=Export in FASTA format
exporter.gff3.description=Export in GFF3 format
exporter.bed.description=Export in BED format
```

Link redirects

You can add a property to WebProperties to have links in results forward the user to a URL instead of the InterMine report page.

config

Add the following to your WebProperties file:

```
# used to redirect links in webapp
webapp.linkRedirect=org.intermine.bio.web.BioLinkRedirectManager

# links to intermine report pages will be replaced with these URLs
externallink.[uniqueID].[class].[taxonId].[field].url = [full URL]
```

uniqueId any string, should be different for each entry, internal use only

class class of Object to redirect, eg. Gene or Protein

taxonId taxon ID or * if config should apply to all

field which identifier field to pass to the URL

full URL full http address, eg. http://www.google.co.uk/#q=monkey

All columns in the results for the configured class will be links to the external URL.

examples

In results pages, all dmel genes will link to FlyBase with Gene.primaryIdentifier instead of the Mine report page:

```
externallink.flybaseResults.Gene.7227.primaryIdentifier.url=http://www.flybase.org/.bin/fbidq.html?<
```

In results pages, all probesets will link to Google instead of the Mine report page:

```
externallink.foo.ProbeSet.*.primaryIdentifier.url=http://www.google.com?q=<<attributeValue>>
```

Next to the value in the column will be a small icon indicating an external link

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 fields to ClassKeys 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 Link 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.

1.6.5 Region Search

BiosegInstallation

You must have *Bioseg* installed to use the region search.

Configuration

struts-config-model.xml

```
<action path="/initGenomicRegionSearchOptions" type="org.intermine.bio.web.struts.GenomicRegionSearch
<action path="/genomicRegionSearch" forward="genomicRegionSearchOptions.page"/>
<action path="/genomicRegionSearchResults" forward="genomicRegionSearchResults.page"/>
<action input="/genomicRegionSearchOptionsBase.jsp" path="/genomicRegionSearchAction" name="genomicRe
  <forward name="genomicRegionSearchResults" path="/genomicRegionSearchResults.do" redirect="false"/>
  <forward name="genomicRegionSearchOptions" path="/genomicRegionSearch.do" redirect="true"/>
</action>
<action path="/genomicRegionSearchAjax" type="org.intermine.bio.web.struts.GenomicRegionSearchAjaxAct
```

tiles-defs-model.xml

```
<definition name="genomicRegionSearchOptions.page" extends="layout.template">
  <put name="body" value="genomicRegionSearchOptions.tile" />
  <put name="pageName" value="genomicRegionSearch" />
</definition>
<definition name="genomicRegionSearchOptions.tile" path="/model/genomicRegionSearchOptionsBase.jsp"
<definition name="genomicRegionSearchResults.page" extends="layout.template">
  <put name="body" value="/model/genomicRegionSearchResultsBase.jsp" />
  <put name="pageName" value="genomicRegionSearchResults" />
</definition>
```

struts-config-form-model.xml

```
<form-bean name="genomicRegionSearchForm" type="org.intermine.bio.web.struts.GenomicRegionSearchForm"
  <form-property name="organism" type="java.lang.String"/>
  <form-property name="featureTypes" type="java.lang.String[]"/>
  <form-property name="pasteInput" type="java.lang.String"/>
  <form-property name="fileInput" type="org.apache.struts.upload.FormFile"/>
  <form-property name="whichInput" type="java.lang.String"/>
  <form-property name="dataFormat" type="java.lang.String" initial="isNotInterBaseCoordinate"/>
```

```
  <form-property name="extendedRegionSize" type="java.lang.String"/>
</form-bean>
```

model.properties

```
genomicRegionSearch.title = Overlap features search from a new list of Genomic Regions
genomicRegionSearch.isNotMultipart = The request is not a file upload request
genomicRegionSearch.spanMisformatted = {0} is in a wrong format
genomicRegionSearch.spanFieldSelection = Please select some {0}
genomicRegionSearch.noSpanPaste = You need to type/paste in some genomic regions
genomicRegionSearch.noSpanFile = You need to type/paste in some genomic regions or select a file to u
genomicRegionSearch.isNotText = {0} is an invalid file type - file must be in plain text format
genomicRegionSearch.noSpanFileOrEmpty = The file you specified does not exist or is empty
genomicRegionSearch.spanInWrongformat = {0} is in a wrong format
genomicRegionSearch.spanInputType = Input type can't be solved
genomicRegionSearch.allRegionInvalid = All regions are invalid. Please do a new search.
genomicRegionSearch.organismEmpty = Organism is empty, please check the data is loaded.

genomicRegionSearchOptions.tab = genomicRegionSearch
genomicRegionSearchOptions.title = Genomic Regions Search
genomicRegionSearchResults.tab = genomicRegionSearch
genomicRegionSearchResults.title = Genomic Regions Search Results

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

web.properties

```
genomicRegionSearch.display = true
genomicRegionSearch.service =
genomicRegionSearch.optionsJavascript =
genomicRegionSearch.resultsJavascript =
genomicRegionSearch.optionsCss =
genomicRegionSearch.resultsCss =
## Make sure pre-defined organisms have chromosome location information in the database
genomicRegionSearch.defaultOrganisms = D. melanogaster
## Exclude feature types for all organisms, comma separated
genomicRegionSearch.featureTypesExcluded.global = GeneFlankingRegion,YouNameItClass
## Exclude feature types for specified organism, semi-colon separated
genomicRegionSearch.featureTypesExcluded.byOrganism = D. melanogaster:GeneFlankingRegion,YouNameItCla
genomicRegionSearch.defaultSpans = 2L:14615455..14619002\\n2R:5866646..5868384\\n3R:2578486..2580016
genomicRegionSearch.caption = Search for features that overlap a list of genome coordinates you ente
genomicRegionSearch.howTo = <ul>\
                              <li>Genome regions in the following formats are accepted:\
                                  <ul>\
                                      <li><b>chromosome:start..end</b>, e.g. <i>2L:11334..12296</i><
                                      <li><b>chromosome:start-end</b>, e.g. <i>2R:5866746-5868284</i>
                                      <li><b>tab delimited</b></li>\
                                  </ul>\
                              <li>Both <b>base coordinate</b> (e.g. BLAST, GFF/GFF3) and <b>interbase
                              <li>Each genome region needs to take a <b>new line</b>.</li>\
                            </ul>
## Query fields when export results as csv/tsv
genomicRegionSearch.query.Gene.views = {0}.primaryIdentifier,{0}.symbol,{0}.chromosomeLocation.locate
genomicRegionSearch.query.Gene.sortOrder = {0}.chromosomeLocation.start asc
```

- Update defaultOrganisms property as needed
- to disable genomic region search, set *genomicRegionSearch.display = false*

- also add *genomicRegionSearch* to *layout.fixed*, e.g.

```
layout.fixed = begin,template,templates,bag,customQuery,query,error,api,genomicRegionSearch
```

- add to ''genomic_precompute.properties'', note: do not duplicate the query number

```
precompute.query.30 = SELECT a3_.shortName AS a1_, a4_.class AS a2_ FROM org.intermine.model.bio.Orga

precompute.query.31 = SELECT a4_.class AS a1_, a5_.name AS a2_, a5_.description AS a3_ FROM org.inter
```

1.6.6 Report page

Report Page

Object Title(s)

One can edit the appearance of object title(s) through the `webconfig-model.xml` file (See Text and messages).

Let us suppose we want to have a default way of displaying bio entities like gene, protein or probe set. Thus we would look up the entry for bio entity class and add the following configuration:

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

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 brackets

What is important to note, is that classes of objects *inherit* from their parents, thus unless we provide a different configuration for a, say, Protein title, the formatting from BioEntity will be applied. Fields resolved in the title(s) will be removed from the summary below it.

Custom Header Link

One can have a custom link in the header of the page through the `webconfig-model.xml` file.

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

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.

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.

Inline Lists

Inline lists are lists of values in one column of a report page table, displayed in one line. Take ''dataSets" on a Gene object as follows:

14 dataSets

name	url
FlyBase data set for Drosophila melanogaster	
Swiss-Prot data set	http://ca.expasy.org/sprot
Drosophila 12 Genomes Consortium homology	
KEGG orthologues data set	
TreeFam data set	http://www.treefam.org/cgi-bin/misc_page.pl?download
GO Annotation from FlyBase	
REDfly Drosophila transcription factor binding sites	http://redfly.ccr.buffalo.edu/?content=/search.php
REDfly Drosophila transcriptional cis-regulatory modules	http://redfly.ccr.buffalo.edu/?content=/search.php
DRSC data set	http://flymai.org/
FlyBase fasta data set for Drosophila melanogaster	

Show more rows

Perhaps we would like to only display the names of data sets available. Then, we would add the following to the Gene class (`<class className="org.intermine.model.bio.Gene">`) in the `webconfig-model.xml` file:

```
<inlinelist>
    <table path="dataSets.name" />
</inlinelist>
```

The result:

14 dataSets

BioGRID interaction data set, IntAct data set, FlyBase data set for Drosophila melanogaster, FlyBase fasta data set for Drosophila melanogaster, Affymetrix array: DrosGenome1, TreeFam data set, REDfly Drosophila transcriptional cis-regulatory modules, GO Annotation from FlyBase, DRSC data set, PubMed to gene mapping, Swiss-Prot data set, REDfly Drosophila transcription factor binding sites, Drosophila 12 Genomes Consortium homology, KEGG orthologues data set

Let's go through the available configuration:

`path` refers to the reference or collection and a key (separated by a dot) that refers to a column (key) we want to display.

`showInHeader` (optional) a boolean attribute that, if set to true, will make the list appear in the header section of the page.

`lineLength` (optional) defines the number of characters we would like to show in the list, any content after this length will be revealed upon clicking an "Expand" link. Bear in mind that the line length will not be exact as it will not split words in half.

`showLinksToObjects` (optional) by specifying this boolean attribute we can create links from the list to the individual objects the list refers to.

If we have not set an inline list to appear in the header section of the page, the list will, by default appear in the category "Other" on the report page. If we login to the system we can then (through the report page tagging interface that is revealed to us) tag the list to appear in a specific section.

Custom Displayers

See Report Displayers for details on how to create displayers for the report page.

Templates

Tag template with the `im:report` tag. See Website Admin.

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 displayer is used

`showImmediately` set to `true` to display the displayer immediately as the page loads, without waiting (`false` by default)

placement the section on the report page the displayer should appear in, can be 'summary' or a valid data category name.

types a comma separated list of class names for this displayer can be used

parameters this is a JSON string used to pass arbitrary parameters to particular displayers, you can make use of this for detailed configuration of any displayers you write. For example, the `trunk/bio/webapp/src/org/intermine/bio/web/displayer/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.

1. Create a Java class [1] in `<mine>/webapp/src/org/intermine/web/displayers` that inherits from `org.intermine.web.displayer.ReportDisplayer`.
2. Implement `public void display(HttpServletRequest request, ReportObject reportObject)` to perform any queries or processing required and put results on the request.
3. Create a JSP file in `<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:

```
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`.

[1] ReportDisplayer makes available a variable called `im` which is the `InterMineAPI` which provides access to config and query execution classes.

SequenceFeature summary

Applicable for any SequenceFeature - shows length, sequence export, chromosome location, cyto location and SO term (where present).

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.SequenceFeatureDisplayer"
    jspName="model/sequenceFeatureDisplayer.jsp"
    replacesFields="chromosome,chromosomeLocation,sequence,length,sequenceOntologyTerm,locations,cyto
    placement="summary"
    types="SequenceFeature"/>
```

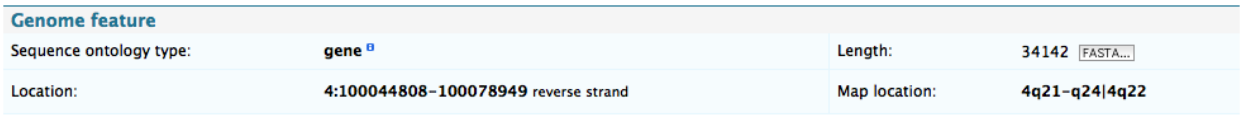


Fig. 1.3: A Sequence feature displayer in metabolicMine.

Protein sequence

Applicable for Protein - shows length, sequence export.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.ProteinSequenceDisplayer"
    jspName="model/proteinSequenceDisplayer.jsp"
    replacesFields="sequence"
    placement="summary"
    types="Protein"/>
```

Fig. 1.4: A Protein sequence displayer in FlyMine.

GBrowse

Show an inline image from a configured GBrowse instance.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.GBrowseDisplayer"
    jspName="model/gbrowseDisplayer.jsp"
    replacesFields=""
    placement="Genes"
    types="SequenceFeature"/>
```

This also needs two properties to be configured in the `minename.properties` file: `gbrowse.prefix` and `gbrowse_image.prefix` which give the location of a running GBrowse instance.

```
gbrowse.prefix=http://www.flymine.org/cgi-bin/gbrowse
gbrowse_image.prefix=http://www.flymine.org/cgi-bin/gbrowse_img
gbrowse.database.source=flymine-31.0
```

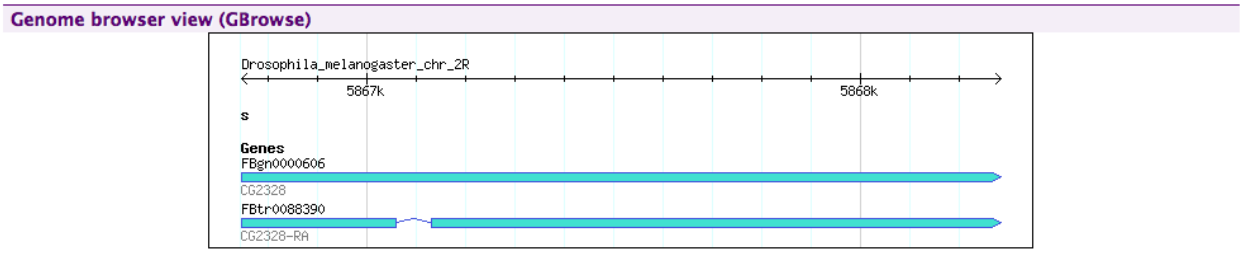


Fig. 1.5: A Genome browser view in FlyMine.

Homologues

Shows a table of organism and homologous genes of homologues per organism.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.HomologueDisplayer"
    jspName="model/homologueDisplayer.jsp"
    replacesFields="homologues"
    placement="Homology"
    parameters="{'dataSets': ['TreeFam data set', 'KEGG orthologues data set']}"
    types="Gene"/>
```

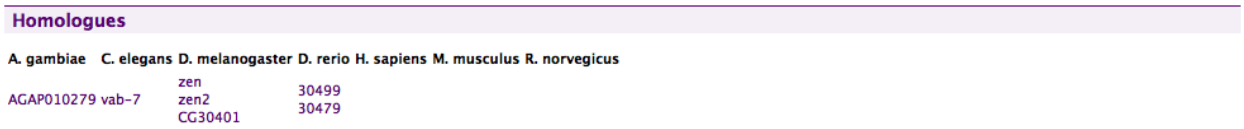


Fig. 1.6: A Homologues displayer in FlyMine.

Note that FlyMine includes a specific displayer to show the twelve Drosophila species as a phylogenetic tree.

Gene structure

Displays transcripts, exons, introns, UTRs and CDSs if present in the model and for the particular organism. Can be added to report pages for any of these feature types and will find the parent gene and show all transcripts, highlighting the feature of the actual report page.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.GeneStructureDisplayer"
    jspName="model/geneStructureDisplayer.jsp"
    replacesFields="transcripts,exons,CDSs,introns,UTRs,fivePrimeUTR,threePrimeUTR"
    placement="Genomics"
    types="Gene,Transcript,Exon,Intron,UTR,CDS"/>
```

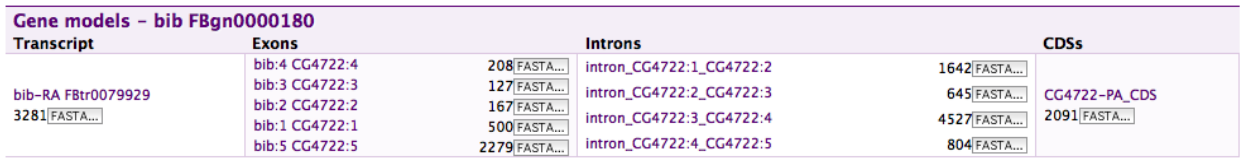


Fig. 1.7: A Gene structure displayer in FlyMine.

Gene Ontology

Simple display of GO terms and evidence codes for a gene, grouped by branch in the ontology. Groups by the three main ontologies (function, process and component) so you may need to run the GO source.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.GeneOntologyDisplayer"
    jspName="model/geneOntologyDisplayer.jsp"
    replacesFields="goAnnotation,ontologyAnnotations"
    placement="Function"
    types="Gene"/>
```

Fig. 1.8: A Gene ontology displayer in modMine.

UniProt comments

A clear view of curated curated comments from UniProt (SwissProt) applied to a protein, or for a gene will show comments from all proteins of the gene.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.UniProtCommentsDisplayer"
    jspName="model/uniProtCommentsDisplayer.jsp"
    replacesFields=""
    placement="summary"
    types="Gene,Protein"/>
```

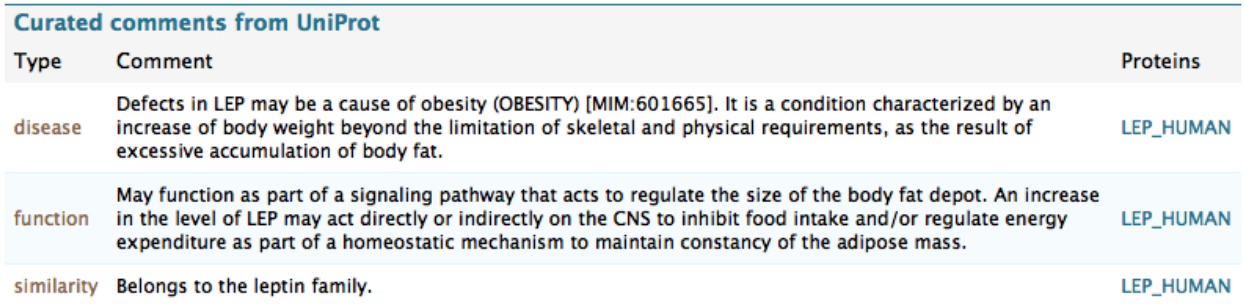


Fig. 1.9: A Uniprot curated comments displayer in metabolicMine.

Interaction network

Uses the Cytoscape Web plugin to display physical and genetics interactions. The interaction displayer links to report pages, allows creation of a gene list of the whole network and can show tabular interaction data. Read NetworkDisplayer for details.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.CytoscapeNetworkDisplayer"
    jspName="model/cytoscapeNetworkDisplayer.jsp"
    replacesFields="interactions"
    placement="Interactions"
    types="Gene,Protein"/>
```

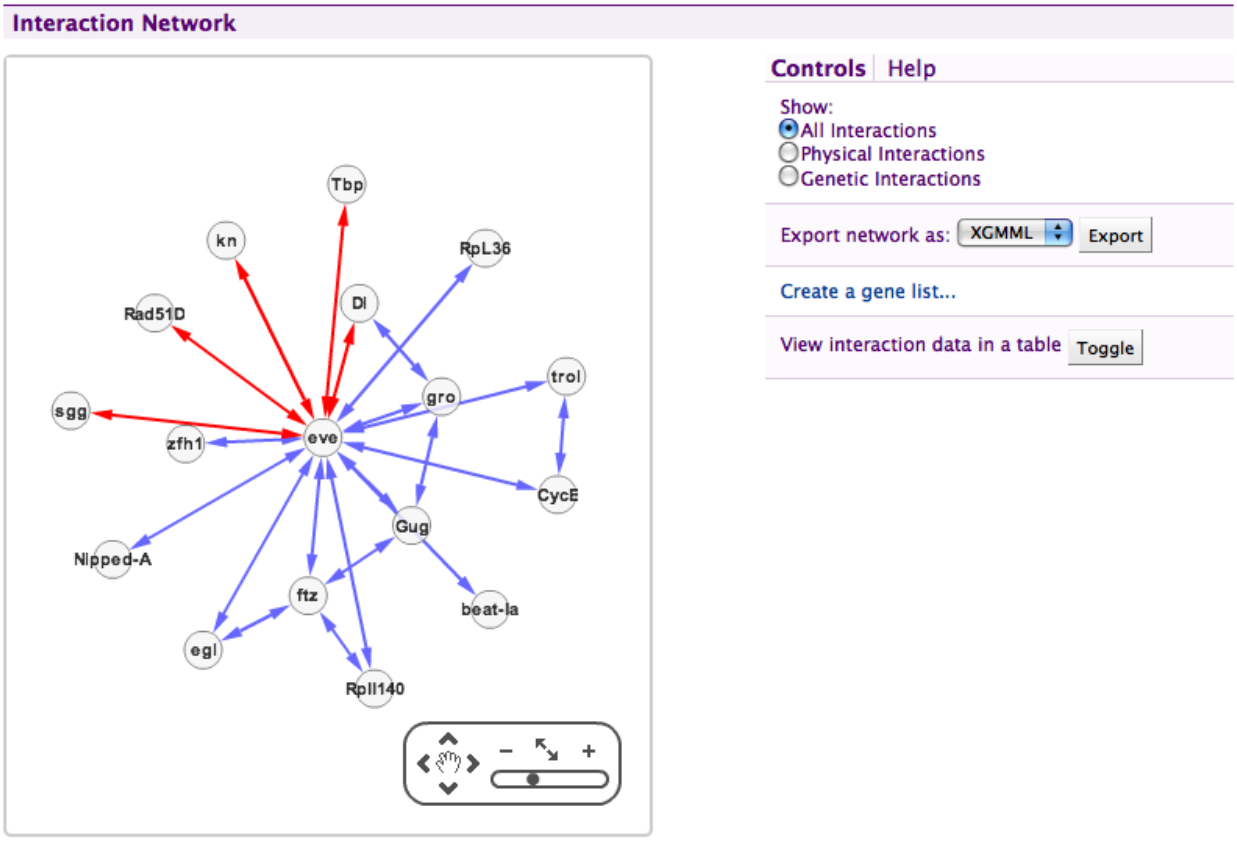


Fig. 1.10: An Interactions displayer in FlyMine.

Overlapping features

A summary view of features that overlap the chromosome location of the reported feature, if the gene structure displayer is also used it will exclude any features that are part of the same gene model - i.e. it won't report that a gene overlaps it's own exons.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.OverlappingFeaturesDisplayer"
    jspName="model/overlappingFeaturesDisplayer.jsp"
    replacesFields="overlappingFeatures"
    placement="Genomics"
    types="SequenceFeature"/>
```

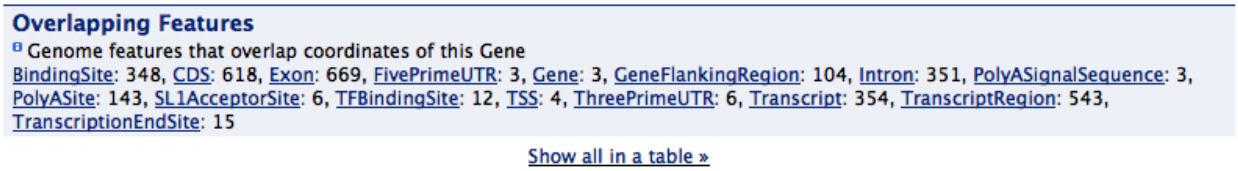


Fig. 1.11: An Overlapping features displayer in modMine.

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.

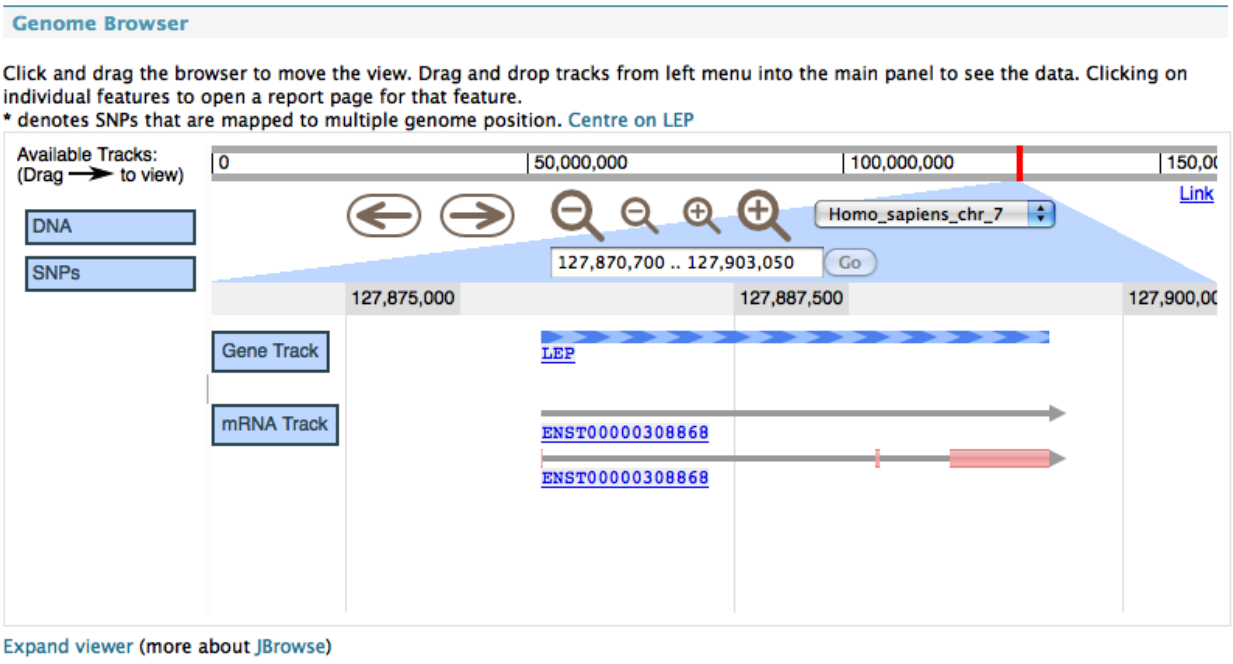


Fig. 1.12: JBrowse genome browser in metabolicMine.

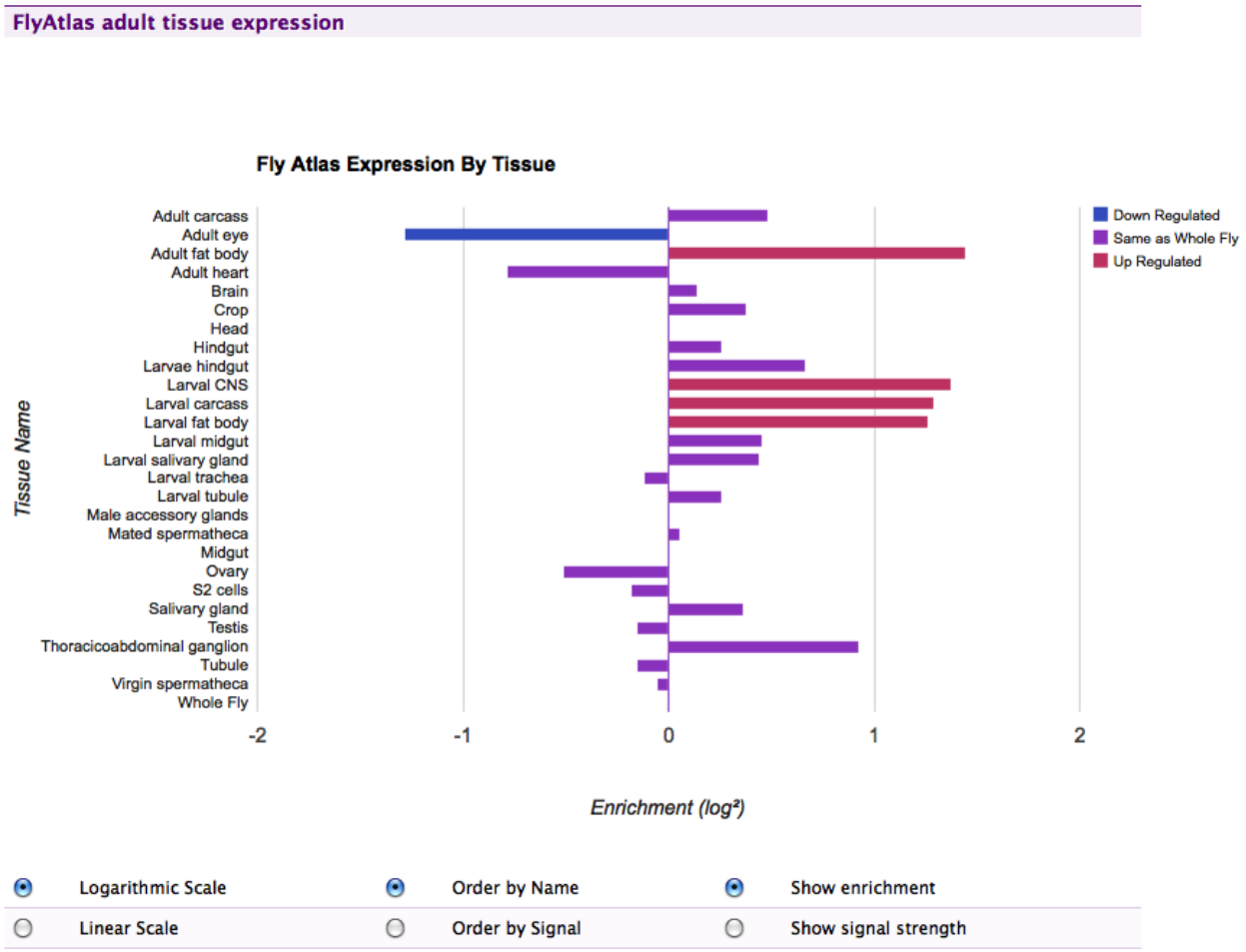


Fig. 1.13: FlyAtlas gene experssion data in FlyMine, this uses the Google Data Vizualization API JavaScript library to render an interactive graph in the browser.

Fig. 1.14: A phylogenetic tree of Drosophila species displayed using the jsPhyloSVG JavaScript library in FlyMine.

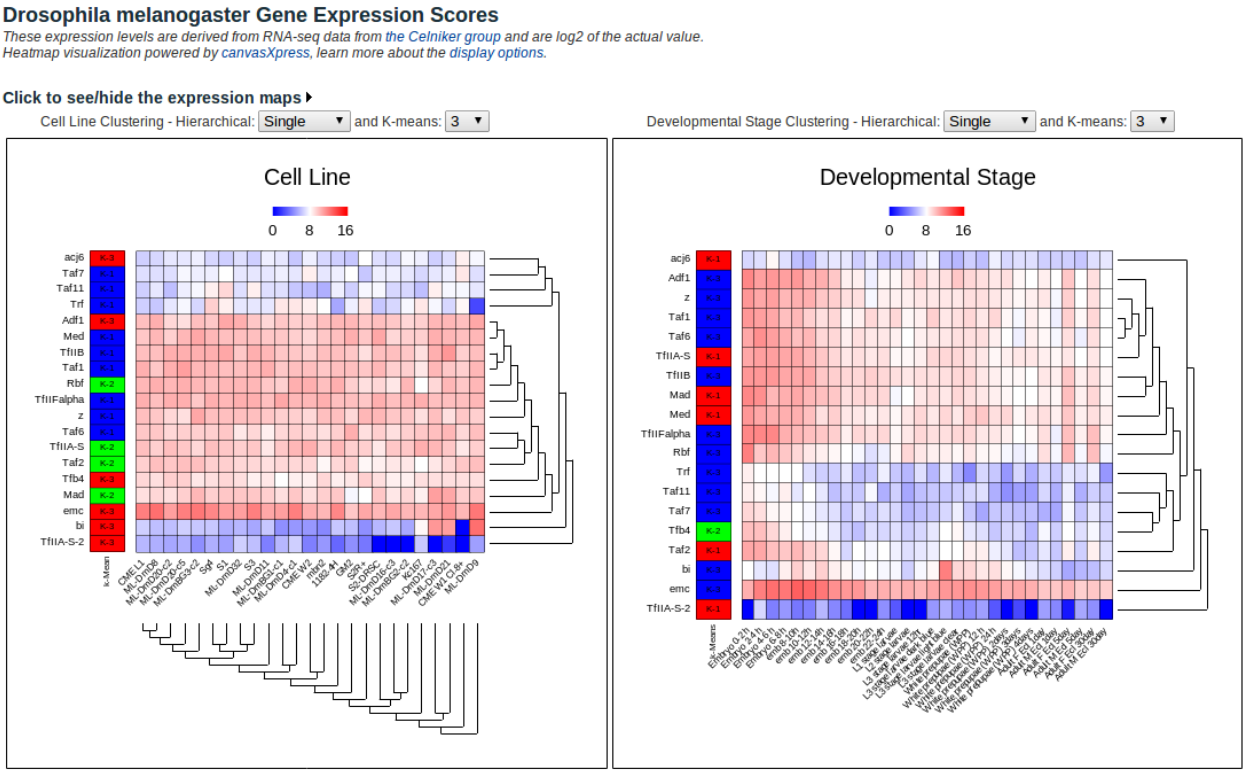


Fig. 1.15: Heatmap of fly gene expression in modMine, this makes use of canvasXpress JavaScript library.

Webapp Tables CSS & HTML

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

CSS	Description
`div.inline-list`	wrapping the list and title in div makes it more clear what elements belong together and allow you to set a custom ID on the whole thing
`div.inline-list h3` (optional)	header 3 (see below) styling
`div.inline-list ul`	list we be displayed inline, without margins between items and without list styles (circles, squares etc.)

0 pathways

Inline List (Inactive, No Results)

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

CSS	Description
`div.inline-list.gray`	one can apply an 'inactive' theme by attaching a class to the top element

0 pathways im:aspect:Function[x] im:summary[x] Add tags

Inline List (Tagging, Right)

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

CSS	Description
`div.inline-list div.right` (optional)	will float element to the right and apply appropriate colors to links; needs to go first, before any other text

synonyms: unnamed, FBgn0014984, l(2)46CFp, l(2)46Ce, V, Group VI, l(2)46CFj, FBgn0015483, evenskipped, Complementation group F, FBgn0019816, Even Skipped, FBgn0019721, VI, FBgn0000606, l(2)46CFh, EVE, l(2)46CFg, evenskip, EVEN-SKIPPED, Group V, 10.5, FBgn0019720, l(2)46Cg, FBan0002328, FBgn0014159, lethal(2)46Ce, FBgn0019718, 14.10, even-skiped, FBgn0023205, FBgn0017400, FBgn0019712, 10.9, F, 20.35, CG2328, E(eve), eve2

'Header' Inline List

```
<div class="inline-list">
    <ul>
        <li><span class="name">synonyms</span>:</li>
        <li>FBgn0014159, </li>
        <li>Complementation group F, </li>
        <li>FBgn0015483, </li>
    </ul>
</div>
```

CSS	Description
`div.inline-list .name` (optional)	the main theme color will be applied to the element

Collection Table

2 clones

secondaryIdentifier	primaryIdentifier
RT01163	FBcl0383804
RT01063	FBcl0383805

```
<div class="collection-table">
    <h3>1 protein</h3>
    <table>
        <thead>
            <tr><th>primaryIdentifier</th><th>primaryAccession</th></tr>
        </thead>
        <tbody>
            <tr>
                <td>EVE_DROME</td>
                <td>P06602</td>
                </tr>
                <tr>
                <td>AUTO_DROME</td>
                <td>P65</td>
            </tr>
        </tbody>
    </table>
</div>
```

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

Note: Modern browsers will apply alternating background and border on odd row columns, the rubbish (IE) will be fixed by running jQuery on page load.

Curated comments from UniProt

Type	Comment	Proteins
function	May play a role in determining neuronal identity. May be directly involved in specifying identity of individual neurons. Pair-rule protein required for segmentation; involved in transforming the broad, spatial, aperiodic expression patterns of the gap genes into a system of precise periodic expression patterns of the pair-rule and segmentary polarity genes.	EVE_DROME
similarity	Belongs to the even-skipped homeobox family.	EVE_DROME
similarity	Contains 1 homeobox DNA-binding domain.	EVE_DROME

Collection Table (Type Column, Text Highlight)

```
<div class="collection-table">
    <h3>1 protein</h3>
    <table>
        <thead>
            <tr><th>primaryIdentifier</th><th>primaryAccession</th></tr>
        </thead>
```

```
        <tbody>
            <tr>
                <td class="class">EVE_DROME</td>
                <td>P06602</td>
                </tr>
                <tr>
                <td class="class">AUTO_DROME</td>
                <td>P65</td>
            </tr>
        </tbody>
    </table>
</div>
```

CSS	Description
`div.collection-table table td.class`	applying a 'class' class will highlight the text in the given column

Genome feature

Sequence ontology type:	**gene**	Length:	1539 FASTA...
Location:	**2R:5866746-5868284** forward strand	Cyto location:	**46C10-46C10**

Collection Table (Vertical Column Border)

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

CSS	Description
`div.collection-table.column-border`	uses a pseudoclass to apply a border between columns

Note: Modern browsers will apply alternating background and border on odd row columns, the rubbish (IE) will be fixed by running jQuery on page load.

Genome feature

Sequence ontology type:	**gene**	Length:	1539 FASTA...
Location:	**2R:5866746-5868284** forward strand	Cyto location:	**46C10-46C10**

Collection Table (Vertical Column Border by 2)

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

CSS	Description
`div.collection-table.column-border-by-2`	uses a pseudoclass to apply a border between every other column

Note: Modern browsers will apply alternating background and border on odd row columns, the rubbish (IE) will be

fixed by running jQuery on page load.

0 miRNAtargets

Collection Table (Inactive, No Results)

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

CSS	Description
`div.collection-table.gray`	one can apply an 'inactive' theme by attaching a class to the top element

131 alleles im:aspect:Function[x] im:hidden[x] im:summary[x] Add tags

primaryIdentifier	symbol	alleleClass	organism.name
FBal0243356	eve[tCH322-103K22]		Drosophila melanogaster
FBal0031221	eve[hs.PS]		Drosophila melanogaster
FBal0039327	eve[2387]		Drosophila melanogaster
FBal0045615	eve[hb.PP]		Drosophila melanogaster
FBal0049482	eve[E+L]		Drosophila melanogaster

Collection Table (Tagging, Right)

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

CSS	Description
`div.collection-table` `div.right` (optional)	will float element to the right and apply appropriate colors to links; needs to go first, before any other text

Collection Table (Persistent Table Headers)

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

CSS	Description
`div.collection-table.persistent`	will make table headers persist as you scroll within the table

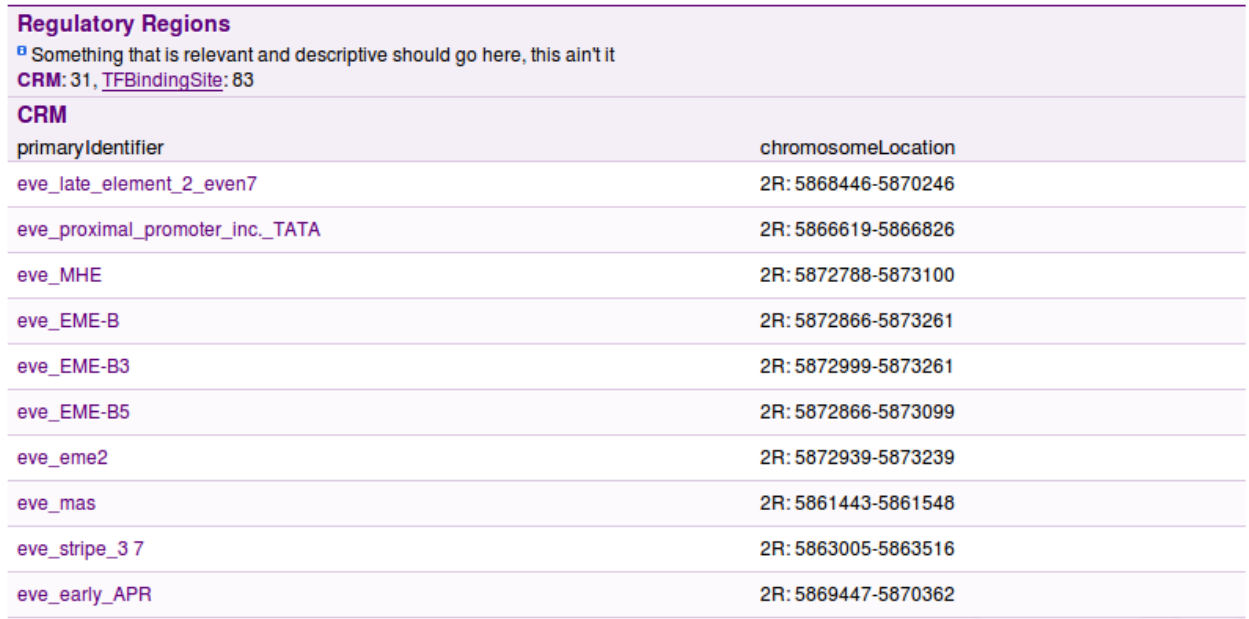
Homologues

A. gambiae	C. elegans	D. melanogaster	D. rerio	H. sapiens	M. musculus	R. norvegicus
AGAP010279	vab-7	zen zen2 CG30401	30499 30479			

Basic Table (Generic)

```
<div class="basic-table">
    <h3>Some title</h3>
    <table>
        <tr><td>Row column</td></tr>
    </table>
</div>
```

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



Collection of Collection Tables

```
<div class="collection-of-collections">
    <div class="header">
        <h3>Regulatory Regions</h3>
        <p>Description</p>
        <div class="switchers">
            <a class="active">CRM</a> <a>TFBindingSite</a>
        </div>
    </div>
    <div class="collection-table">
        <-- ... -->
    </div>
    <div class="collection-table">
        <-- ... -->
    </div>
</div>
```

CSS	Description
`div.collection-of-collections`	a div wrapper for collections
`div.collection-of-collections div.header`	will apply a background color that of collection table header
`div.collection-of-collections div.header a.active` (optional)	link elements are underlined by default and switched to bold if class 'active' is applied to them

	1994	DNA Seq	4	347-54	7841458
	2001	Bioessays	23	698-707	11494318
osition-specific mechanisms.	2002	Development	129	4931-40	12397102

Show more rows Collapse

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

```
<div class="collection-table">
    <-- ... -->
    <div class="toggle">
        <a class="less">Show less</a>
        <a class="more">Show more</a>
    </div>
    <div class="show-in-table">
        <a href="#">Show all in a table</a>
    </div>
</div>
```

CSS	Description
`div.collection-table div.toggle a.more`	will create apply an expand/more button
`div.collection-table div.toggle a.less`	will create apply a collapse/less button; bear in mind that if you want to show it to the right like on report pages, it needs to go before other toggles and be floated right
`div.collection-table div.toggle a` (optional)	a generic button without any upward/downward arrows
`div.collection-table div.show-in-table a`	the appropriate color will be applied to the link contained, no more, no less (in fact, show all)

Title (Level 3)

Link to other InterMines

```
<h3 class="goog">Link to other InterMines</h3>
```

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

Smallfont, Display one-per-line

Homologues

A. gambiae	C. elegans	D. melanogaster	D. rerio	H. sapiens	M. musculus	R. norvegicus
AGAP010279	vab-7	zen zen2 CG30401	30499 30479			

```
<table class="tiny-font">
    <tr><td class="one-line">
        <a>One</a>
```

```
        <a>Two</a>
    </td></tr>
</table>
```

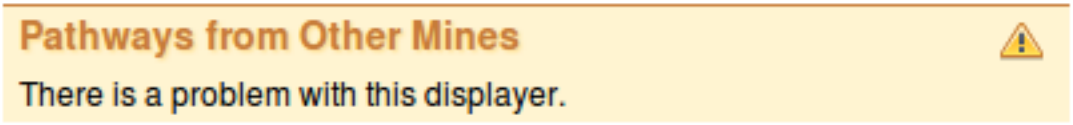
CSS	Description
`.one-line *`	applying class 'oneline' will make *all descendants* appear one per line
`.tiny-font *`	will apply 11px font size to *all descendants*

Loading spinner (AJAX)

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

CSS	Description
`.loading-spinner`	will show an inline block positioned loading spinner gif

Table Warning




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

CSS	Description
`.warning`	will show a warning icon and change the color of the div to pale yellow

Report Widgets

New in version 1.1.

Note: Client side development is the best thing since sliced bread but for a novice the following *will* be overwhelming at first. We are learning too.

Report Widgets are a type of embeddable component that offers an unparalleled **flexibility** when it comes to the tools that InterMine offers. They are provided as a way for developers to create interactive *widgets* to their end users. Such widgets can then be embedded on virtually any page, be it within a mine or outside of it.

See also:

Report widgets demo.

This document describes the steps needed to create an example publications widget that lists publications related to a particular gene.

The steps outlined:

1. Install dependencies for the widget precompiler.
2. Develop widget consisting of **behavior** classes, **styles** and **templates**

250 Publications for zen

Page: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Title	Author
Molecular and genetic organization of the antennapedia gene complex of Drosophila melanogaster.	Kaufman TC, Seeger MA, Olsen G
Role of the zerknüllt gene in dorsal-ventral pattern formation in Drosophila.	Levine M, Rushlow C
Conservation and divergence in molecular mechanisms of axis formation.	Patel NH, Lall S
Signaling pathways that establish the dorsal-ventral pattern of the Drosophila embryo.	Anderson KV, Morisato D
Identifying spatially similar gene expression patterns in early stage fruit fly embryo images: binary feature versus invariant moment digital representations.	Kumar S, Gurunathan R, Panchanathan S, Van Emden B
Fast sequence evolution of Hox and Hox-derived genes in the genus Drosophila.	Ruiz A, Barbadilla A, Negre B, Casillas S
Transcriptional regulators of Drosophila embryogenesis.	Manley JL, Read D
The establishment and interpretation of transcription factor gradients in the Drosophila embryo.	Courey AJ, Huang JD
Drosophila: flying high in Zürich.	Wilkins AS
A cluster of Drosophila homeobox genes involved in mesoderm differentiation programs.	Jagla K, Frasch M, Bellard M

Fig. 1.16: Publications for zen gene coming from FlyMine.

3. Provide config for the widget
4. Run it

Install dependencies

Node.js The system that precompiles widgets is based on the Node platform which means one can write applications in JavaScript that run on top of Google's V8 engine. Head over to the homepage or visit this Joyent guide to install the platform. Remember that installing by building will execute built in tests that can take a couple of minutes to run on slower machines.

Make sure that you have the platform's installer npm installed. It is used to download packages that you can then reference in your Node apps.

Note on CoffeeScript Although JavaScript is the main language of the Node platform we will be using CoffeeScript which is a little language that compiles to JavaScript. The reason is that the code is more readable and one can use *nice* syntax when, for example, working with arrays.

Reference precompiler Now head over to the GitHub repo of a reference client/precompiler. This tool will allow you to quickly iterate over your code and test it in the browser immediately.

Typical steps (refer to the repo's docs) will look as follows:

```
$ git clone git@github.com:radekstepan/intermine-report-widgets.git
$ cd intermine-report-widgets/
$ npm install -d
$ npm start
```

Now you should have a running instance of a service providing widgets. Head over to http://127.0.0.1:1119 to confirm that it is indeed the case.

Develop a widget

Create a new folder (no spaces and special characters in the name) in the `widgets/` folder of the precompiler.

Presenter The next step is writing a presenter which is a component that knows how to get data for itself and then render them in a particular way, thus it encapsulates the behavior of the widget.

The file needs to be called `presenter.coffee` and be placed in a directory with the name of the widget. The file needs to contain a `class Widget` with the following signature:

```
class Widget

    # Have access to config and templates compiled in.
    constructor: (config, templates) ->

    # Render accepts a target to draw results into.
    render: (target) ->
```

The constructor is passed two variables, `config` and `templates` which are objects that have references to config from a mine and templates that we will write in the next step.

The `render` function is passed a target variable which is the element where we will want to place the widget's output.

It is up to you what you do in between. The reference implementation of the precompile containes different widgets that you can inspect and see how they work. Many of them use a client side framework called Backbone to efficiently manipulate data and display them. It is up to you which framework you use, if any. In the last step, config, we will learn how to setup the widget to download these libraries for us when needed.

Templates Templates represent pieces of reusable HTML that can be filled with data from a widget. We create one by creating a file ending with `.eco` in the widget's directory. The language of choice here is eco. It allows us to use CoffeeScript syntax inside HTML. You can think of JavaServerPages as being a Java equivalent.

Inside the widget, templates are precompiled into a function form so that if you want to pass a bunch of variables into a template and make it into HTML do as follows:

```
templates['myTemplateName']
    'foo': 'This is some text'
```

Styles Styles or CascadingStyleSheets inside widgets have the advantage of applying only to the widget itself and not any other content on the page. So for example the following:

```
h4 { color:red; }
```

Will make all Header4 elements in the widget red, not any other Header4 elements on the page. Thus you can reasonably safely embed the widget on any page you want. Although, be aware of the fact that many pre-existing styles on the page may not play as nice and may color/change your widget if its rules are too generic and broad.

Top create a style create a new file `style.css` in the directory where the widget resides and write some style rules.

Stylus You do not have to stick to plain old CSS and can use the Stylus language instead. It is a CSS preprocessor just like CoffeeScript is a JavaScript preprocessor. And just like it it makes CSS better by adding variables, nested rules, calculations etc. At least one example widget should contain a Stylus style.

To create a Stylus stylesheet instead of a CSS one, simply create the file `style.styl` instead of `style.css` and write your rules there.

Config

Now that we have coded our widget, we need to configure it. This is done in the `config.json` file of the reference implementation service.

One wants to add a new key value pair into the `widgets` value with the key referring to the name of the widget/name of the folder where the widget resides. So if our widget is called `emo-widget`, our config file will look like this:

```
{
    "service": {
        "port": 1119
    },
    "widgets": {
        "emo-widget": {}
    }
}
```

Inside the widget config add the following key value pairs:

- **author** referring to the author of the widget
- **title** is a snappy oneliner representing the widget
- **description** is less snappy than the above
- **version** allows one to keep track of versions of a widget

These are useful as when we inspect a widget we can see what its intention is, when it was made etc.

Then there are two directives that define libraries (JavaScript/CSS) to load for the widget and one to define what values will be passed on/embedded in the widget.

Dependencies To define library dependencies of a widget, use the `dependencies` key pointing to a list. You can even specify if we need to first wait to fetch a library before fetching another (synchronous loading). In the following example, we fetch 3 libraries. The first two are JavaScript files where the second (and subsequent) one waits for the first one to finish loading. The last library asks for a CSS file. Bear in mind that all files are included on the page without any prefixes. So you need to deal with a potential that two libraries will not work well together and styles are clashing.

```
"dependencies": [
    {
        "name": "jQuery",
        "path": "http://somwhere/jquery.js",
        "type": "js",
        "wait": true
    },
    {
        "name": "_",
        "path": "http://somewhere/underscore.js",
        "type": "js"
    },
    {
        "path": "http://somewhere/style.css",
        "type": "css"
    }
]
```

Config Then we have a dictionary under the `config` key. You can use it to pass specific config to a widget. The idea is that you write your widget to be quite generic and then use the config to make it specific. The dictionary accepts

key value pairs that are up to you to choose, the only exception is the `pathQueries` key that points to a dictionary of mine's PathQueries. Use JSON syntax to write such queries, not XML.

In the following example we specify the mine's address and a partial PathQuery to run.

```
"config": {
    "mine": "http://yeastmine-test.yeastgenome.org:8080/yeastmine-dev/",
    "pathQueries": {
        "expressionScores": {
            "select": [
                "TYPE.expressionScores.score"
            ]
        }
    }
}
```

Run it

To run the widgets, you need to include InterMine's API loader that is used for loading widgets. The easiest way to get it is by including the following JavaScript file on a page:

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

Now we say that we want to load report widgets passing in a callback function. In this callback we specify that we want a new ReportWidgets instance pointing to a service serving them.

```
intermine.load('reportWidgets', function() {
    var widgets = new intermine.reportWidgets('http://127.0.0.1:1119');
});
```

In this callback still we say which widget we want passing in extra config that should be merged with service config. This way we can pass in say a symbol of a specific gene we have on a 'page'.

```
widgets.load('spell-histogram', '#spell', { 'type': 'Gene', 'symbol': 'S000001863' });
```

Workflow

blue represents a common workflow to load widget loaders be it for report or list analysis widgets.

purple represents a flow of List Widget Loader asking the mine for JSON results for a specific type of a widget.

green represents a flow of Report Widget Loader asking for a JS of a specific widget.

Requirements

Service

1. Compile **templates** into their JS form and make them accessible within the context of the widget only.
2. Make *CSS* available only in the context of the widget, perhaps by prefixing each declaration with a dynamic `widget id` using prefix-css-node or css-prefix.
3. Respond to the client with a list of **resources** that need to be loaded beforing rendering the widget.
4. Each widget consists of:
   (a) One CoffeeScript **presenter** containing the logic getting data from the **model** using imjs.

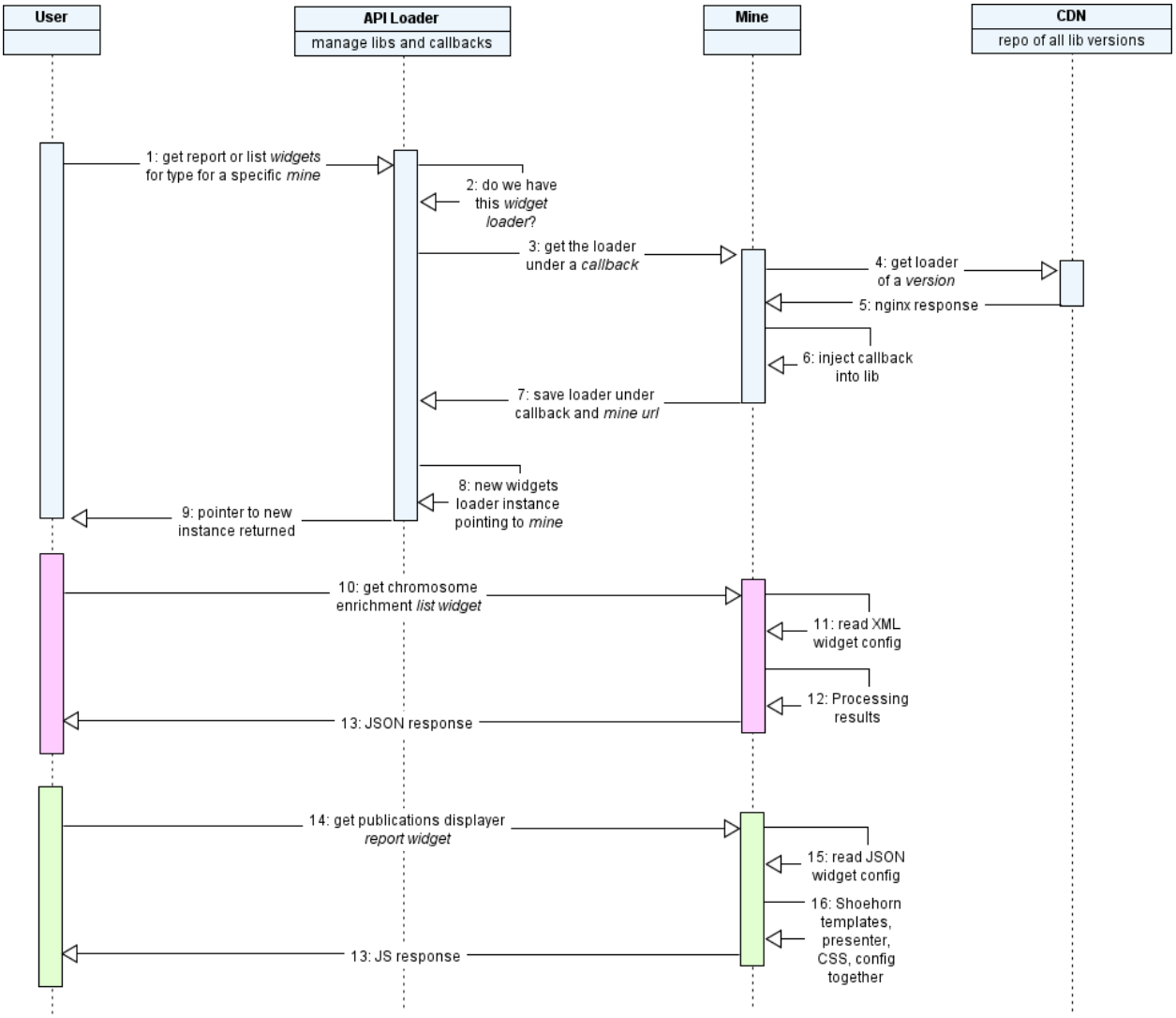


Fig. 1.17: A workflow of how list and report widgets are processed client and server side.

   (b) A number of eco **templates** precompiled.

   (c) One **CSS** file specifically for the widget.

   (d) Any extra **config** dynamically populated for the widget to consume. This could be the mine the Widget is to take data from or extra flags that specialize an otherwise generic Widget.

   (e) Optional number of requirements (CSS, JS), loaded from the CDN.

5. All of the previous are configured by the user and the service validates that all widgets are executable.
6. **Data** requests are done from within the widget to speed up their initial loading.
7. Files are served as UTF-8.
8. Provide nice URL for fetching the widgets so it is easier to debug them in Network view, `/widget/24517/publications-displayer`.
9. Provide info messages on each step of the compilation process so we can determine where problems lie. These then be returned as *message* to the user when requesting widgets as HTTP 500 JSON errors.

Optional

- Cache resources by, for example, not packaging resources on the fly but doing so on service startup. Then, say the latest modification date. Add `ETag` and return `304` not modified then.
- Allow the use of LESS instead of CSS.
- Allow the use of other templating languages.
- Check for the presence of `Displayer.prototype.render` and `Displayer.prototype.initialize` in the compiled **presenter**.
- Validate that callbacks are valid JavaScript identifiers. Should not be needed as we will use API loader and generate these automagically.
- Provide a signature in the generated output describing the title, author etc for the widget in question.
- Each block in the compiled result have a comment header so it is easier to find where things lie when debugging.
- Provide connection to imjs by default.

Issues

- If we want to split presenter across multiple CoffeScript files, how to maintain their order in the resulting JS version? Go alphabetically?

Client

1. Make use of intermine-api-loader to efficiently load resources and libs only when needed.
2. Generate **callbacks** that are unique for the page taking into account other clients that could exist on the page. As the service URL is unique per client, make use of that.
3. Dump error messages from the server into the target element where widget was supposed to have been.
4. Cache all of the widgets listing as we need to be resolving widget dependencies first.
5. Provide a wrapping `<article>` element with a predictable `im-report-widget` class so we can use it in our CSS.

Optional

- Provide a callback where all widgets can dump error messages.

1.6.7 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 *objectstoresummary.config.properties*

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.

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
```

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 all fields you want to be autocompleted to this file, like so:

```
# in MINE_NAME/dbmodel/resources/objectstoresummary.config.properties
org.intermine.model.bio.Disease.autocomplete = description
```

2. Add the postprocess to your *MINE_NAME/project.xml* file.

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

3. 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 defaul

Templates page

To have templates appear on the templates page, create a template as a SuperUser and tag the template with the "im:public" tag.

The templates are sorted by most popular first. If the user is logged in the user's most popular templates are shown first.

1.6.8 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.

```
help.page.<parsed webpage name> = <anchor in help.html file>

help.page.begin=begin
help.page.templates=templates
help.page.bag=lists
help.page.bag.upload=lists:upload
help.page.bag.view=lists:view
help.page.customQuery=customQuery
help.page.mymine.lists=mymine:lists
help.page.mymine.history=mymine:queryHistory
help.page.mymine.saved=mymine:savedQueries
help.page.mymine.templates=mymine:savedTemplates
help.page.mymine.password=mymine:changePassword
help.page.dataCategories=data
help.page.objectDetails=reportPage
help.page.template=template
help.page.results=results
help.page.bagDetails=listAnalysis
help.page.bagUploadConfirm=buildList
help.page.query=query
help.page.importQueries=importQueries
help.page.importTemplates=importTemplates
help.page.tree=tree
help.page.aspect=dataCategory
```

Your mine's web.properties file is merged with this web.properties file, so entries you add to web.properties will overwrite the values listed above.

Data definitions

Update these in the classDescriptions.properties file.

1.6.9 QueryBuilder

Select a Data Type to Begin a Query

types in bold Tag types with *im:preferredBagType* tag. Use the model browser to tag classes, eg. http://www.flymine.org/query/tree.do

intro text Most text in InterMine can be set in model.properties, see Text and messages.

help text Set in *classDecriptions.properties* file

query builder

SUMMARY Which columns appear when you click on SUMMARY button are set in WebConfigModel.

autocomplete Add fields to the ObjectStoreSummaryProperties file to have their form fields autocomplete.

1.6.10 Website Admin

1.6.11 Superuser

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 MineManager's Mine Settings > Web screen.

Templates

All logged in users can create template queries, but the SuperUser can make them available to all users by tagging them as public templates. Making a template query is an easy way to get users of your webapp to the data they want very quickly.

Tagging

Template queries and lists

The SuperUser can change where templates and lists appear by adding tags via the templates and lists pages in the MyMine section of the webapp. Only the administrator can apply/view/edit tags starting at *im:* The tag data is stored in the user-profile database.

tag	purpose
im:public	make list/template viewable by all users
im:frontpage	put list/template on home page
im:converter	template used in generating links in the 'Convert' section on the list analysis page
im:aspect:CategoryName	template appears underneath specified category. For instance template with im:aspect:Genomics tag will be displayed in Genomics category on the report page
im:report	allows template to be displayed on report or list analysis page
im:admin	prevents template from being displayed on templates page
im:order:n	specify the order lists should go in (on homepage only currently). If two lists have the same Integer "n" value, natural ordering on the list name will be applied as a decisive criterion

Fields and collections

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

tag	purpose
im:hidden	hides the field/collection
im:summary	add collection to 'Summary' section of report page
im:aspect:CategoryName	collection appears underneath category

Classes

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

tag	purpose
im:aspect:CategoryName	class appears on aspect page
im:preferredBagType	class appears first in the class selection

im:converter tag

If a template is tagged with *im:converter*, it is:

1. Used by the list analysis page, in the "Convert" section.
2. Used by the list upload page to converter between types.

- Eg, the user pastes in a protein identifier, but chooses "Gene" from the type drop down menu. A converter template can be used to look up the *Gene* corresponding to the given *Protein*.

To work as a converter the template must follow the following pattern:

- the top-level class in the query must be the class we wish to convert *from* (eg. *Gene*)
- there must be exactly one editable constraint - the *id* field of the top level class (eg. *Gene.id*)
- the fields selected for output must be *Gene.id* and the id field of the class to convert *to*

Normally the *id* field isn't shown in the query builder and probably isn't useful in other queries. Only the administrator user can create queries using the *id* field. Here is an example converter template:

```
<template name="Gene_To_Protein_Type_Converter" title="Gene to protein type converter" longDescriptio
        <query name="Gene_To_Protein_Type_Converter" model="genomic" view="Gene.id Gene.proteins.id"
        <node path="Gene" type="Gene"></node>
        <node path="Gene.id" type="Integer">
                <constraint op="=" value="0" description="Gene.id" identifier="Gene.id" editable="tru
        </node>
```

```
        </query>
</template>
```

1.6.12 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 porduct
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_current* 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 Google and Yahoo.

To sign in with one of these authentication providers: 1. Click on ‘‘login’‘ (in the upper-right). 1. Click the name of the Open-ID provider you wish to use. 1. Authenticate yourself with your provider. 1. You will be redirected to your mine when finished.

To set this up for a mine you administer:

- The most important thing is to set up a couple of properties correctly in your mine's properties file (located in the *.intermine* directory), eg:

```
webapp.baseurl=http://beta.flymine.org
webapp.path=intermine-test
```

If you do not wish to allow Open-ID accounts, set the property “openid.allowed=false” in any of the property files that end up in the WEB_PROPERTIES map.

1.6.13 Layout

Note: This text describes how to customize the look & feel of the whole app.

Parts

Logo

The logo is independent from any themes and is located here `<your_mine>/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`

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*

```
<li id="api"  <c:if test="${tab == 'api'}">class="activelink"</c:if>>
  <a href="/${WEB_PROPERTIES['webapp.path']}/api.do">
    <fmt:message key="menu.api"/>
  </a>
</li>
```

Then add the text for that tab to your *MINE_NAME/webapp/resources/model.properties* file:

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

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

```
# in MINE_NAME/webapp/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

Favicon

Favicon (icon seen next to the url of the webapp in the browser url bar) can be set by adding the following line:

```
<link rel="shortcut icon" type="image/x-icon" href="model/images/favicon.ico">
```

Into the `webapp/resources/webapp/layout.jsp` file and its `</head>` section. The favicon itself should be located in `<your_mine>/webapp/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:

```
# 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:

```
# 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.

1.6.14 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

Query XML

Links to results of query. Can run any query built by QueryBuilder; QueryBuilder generates the XML.

http://www.flymine.org/query/loadQuery.do?skipBuilder=true&query=%0A%3Cquery+name%3D%22%22+model%3D%22genomic%

Link to List Analysis page

Template results

Links to list analysis page comprised of results of template query. "path" attribute determines which column used to create list. URL available on template form in webapp.

http://www.flymine.org/query/loadTemplate.do?name=Pathway_Genes&constraint1=Pathway.name&op1=eq&value1=Pentose+phosph

List of Identifiers

Links to list analysis page for specified objects. For a very long list, use a form instead of a link. Can use any identifiers.

http://www.flymine.org/query/portal.do?externalids=CG2262,CG3069,CG2859,CG5041,FBgn0036513&class=Gene

Report page

Links directly to report page. URL available on report page in webapp.

http://www.flymine.org/query/portal.do?externalid=FBgn0004053&class=Gene&origin=flybase

Link into Mine with Orthologues

The example URL contains ''D. melanogaster" genes. The results will contain the corresponding ''C. elegans" genes, if any. This will only work if you have orthologue data loaded into your Mine. Will forward to report page OR list analysis page.

http://www.flymine.org/query/portal.do?externalids=CG2262,CG3069,CG2859,CG5041,FBgn0036513&class=Gene&orthologue=C.%

Convert any identifiers to Genes

When linking to a report page or a list analysis page you can convert the data type, for instance if you provide a Protein identifier and want to link to the corresponding Gene, you need to specify the class as Gene. Will only work if you have a converter template available

http://www.flymine.org/query/portal.do?externalid=EVE_DROME&class=Gene

More examples

See FlyMine for more examples: http://blog.flymine.org/link-to-flymine

1.6.15 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*

1.6.16 Third party tools

Galaxy

Enable Galaxy export

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

```
# 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/><br/>You can run queries by click
```

Update Struts config

```
# 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"/>
```

```
# 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.do"/>
```

```
# MINE/webapp/resources/struts-config-model-form.xml
<form-bean name="galaxyExportForm" type="org.intermine.bio.web.struts.GalaxyExportForm"/>
```

Customization

Properties located in the ‘’web.properties’‘ file.

parameter	purpose	required?
display	enable Galaxy export	yes[1]_.
disabledMessage	displayed when Galaxy export is disabled	yes
baseurl.default	base url of Galaxy server	yes[2]_.
url.value	tool runner url	yes[3]_.
welcomeMessage	displays on the homepage when coming from Galaxy	yes

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

GBrowse

Link out to another GBrowse

webapp.viewByID.prefix portal.do?externalid=

gbrowse.config.directory SERVER_NAME:/gbrowse/conf/

gbrowse.www.directory SERVER_NAME:/gbrowse/www/browser/gbrowse

gbrowse.prefix http://www.flymine.org/cgi-bin/gbrowse

gbrowse_image.prefix http://www.flymine.org/cgi-bin/gbrowse_img

GBrowse Instance for your mine

GBrowse is an interactive, generic, web-based genome annotation viewer created as part of the GMOD project. InterMine is able to export GFF3 and FASTA files suitable for importing into a GBrowse instance. Links in GBrowse can then refer to InterMine object report pages and the report pages can have links to GBrowse.

The GBrowse website has installation instructions and a tutorial. BioPerl is a requirement of GBrowse.

Properties The dbname, username and password are set in your mine properties file:

```
project.title=FlyMine
project.sitePrefix=/flymine
gbrowse.database=gbrowse
gbrowse.database.adaptor=mysql
gbrowse.database.source=my-mine-release-1.0
gbrowse.database.host=localhost
gbrowse.database.user=username
gbrowse.database.password=password
gbrowse.database.write.user=username
gbrowse.database.write.password=password
```

The *gbrowse.database.source* property is user visible name of the GBrowse configuration. It will appear in URLs and on the GBrowse pages. For example in FlyMine a GBrowse URL could be:

```
http://www.flymine.org/cgi-bin/gbrowse/flymine-release-11.0/?label=Genes;name=FlyMineInternalID_1090
```

In this case *flymine-release-11.0* is the *gbrowse.database.source*. This property is also used by GBrowse to choose the configuration file to use from the *gbrowse.config.directory* (see below). For example if the property is set to *test-mine*, the file will be called *test-mine.conf*.

The gbrowse.database.write.*, gbrowse.database and gbrowse.database.host properties are used by the *load-gbrowse-data-mysql* target. The gbrowse.database.user and gbrowse.database.password, gbrowse.database.host, gbrowse.database and gbrowse.database.source properties are used by *ant install-gbrowse-conf* (see below).

Add GBrowse project Add */gbrowse* directory to your mine. You can copy FlyMine's, simple replace all mentions of FlyMine with the name of your mine.

Creating files for GBrowse On your local machine, in *MINE_NAME/gbrowse*:

```
$ ant write-gff
```

which writes GFF and FASTA files for each chromosome to the *MINE_NAME/gbrowse/build/gbrowse/data/* directory.

This command may use more than the default memory available for Java. See the note about setting *ANT_OPTS*, at the bottom the [wiki:Prerequisites] page to increase the limit.

Loading into GBrowse To load GFF and FASTA files into a local GBrowse schema you need:

- a machine with GBrowse, BioPerl and MySQL installed
- a 'gbrowse' database present in MySQL (use: *mysqladmin create gbrowse*)
- set your umask to 0002 (to create files readable to all), which is needed so the the MySQL server can read the files created by the next step

```
$ umask 0002
$ ant load-gbrowse-data-mysql
```

This loads the GFF and FASTA files into a local (MySQL) gbrowse schema.

If there are problems, run *ant* with the *-v* flag.

Installing the MINE_NAME.conf file First make sure that these two properties are in your *MINE_NAME.properties* file:

```
gbrowse.config.directory=/etc/apache2/gbrowse.conf
gbrowse.www.directory=/var/www/gbrowse
```

The directories will vary depending on where GBrowse is installed. You can also use any valid *rsync* path, eg. *some_user@some_machine:/some/path/*

Run this in *MINE_NAME/gbrowse*:

```
$ ant install-gbrowse-conf
```

which command copies the a GBrowse configuration file from *MINE_NAME/gbrowse/resources/MINE_NAME.conf* into Apache's GBrowse conf directory (as configured with the *gbrowse.config.directory* property). While being copied the filename is changed to <*source_name*>*.conf* where <*source_name*> is the value of the *gbrowse.database.source* property.

You will need to make sure that the *gbrowse.config.directory* is writable by the user.

Databse host, username and password are also read from the properties and substituted into *<source_name>.conf*.

GBrowse uses the filename of the configuration file to create the URL to access. eg. setting gbrowse.database.source to flymine-release-3.0 gives a database that can be accessed at: http://www.flymine.org/browser/bin/gbrowse/flymine-release-3.0

Cytoscape network viewer

The network viewer is based on the CytoscapeWeb flash component.

The current working example is gene interaction displayer, it links to gene/protein report pages, and displays the gene on the report page, its interacting genes, and physical and genetics interactions between them.

Configuration

1. add the following entry to your ‘‘webconfig-model.xml’‘ file:

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.CytoscapeNetworkDisplayer"
        jspName="model/cytoscapeNetworkDisplayer.jsp"
        replacesFields="interactions"
        placement="Interactions"
        types="Gene,Protein"/>
```

2. add the following entries to your ‘‘struts-config-model.xml’‘ file:

```
<action path="/cytoscapeNetworkExport" type="org.intermine.bio.web.struts.CytoscapeNetworkExportActi
<action path="/cytoscapeNetworkAjax" type="org.intermine.bio.web.struts.CytoscapeNetworkAjaxAction"/>
```

3. re-release your webapp and you should see the interaction displayer on gene report pages.

Data format

CytoscapeWeb can parse the following date formats:

- SIF
- XGMML
- JSON

Network can be exported as:

- SIF
- XGMML
- PNG
- SVG
- TSV
- CSV

Implementation

What we do for the interactions viewer (the one that targetmine has) is generate the query in Java, e.g.:

```
select * from interactions, genes where gene = [the gene on the report page]
```

This query returns all the interactions for that gene of interest. Then we do another query to find out how the genes in the interactions are related. These data are then sent to the Cytoscape viewer via ajax call in SIF/XGMML/JSON format, and the Cytoscape viewer displays those data.

Table of some source files and their functions:

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

CytoscapeNetworkAjaxAction.java the Structs action class (config in struts-config-model.xml) to handle the http request and call CytoscapeNetworkService to generate the network data

CytoscapeNetworkGenerator.java generate network data in SIF/XGMML/JSON format

CytoscapeNetworkService.java service class

CytoscapeNetworkNodeData.java netowrk work node model

CytoscapeNetworkEdgeData.java netowrk work edge model

After fetching the data to the web page, you can customise the outlook of the network by using CytoscapeWeb javascript API.

Heatmap

InterMine makes use of canvasXpress heatmap to visualize gene expression data.

CanvasXpress is a javascript library based on the <*canvas*> tag implemented in HTML5. It is written by Isaac Neuhausi.

Hierarchical and k-Means clustering algorithms and zoom in/out functionality have been implemented within the heatmap.

An example in modMine

A specific heatmap application can be referred in modMine. It visualizes fly expression data (example) generated from modENCODE project.

The raw data is parsed and converted to InterMine objects. In a Struts controller, the expression data will be fetched by running a InterMine path query and parsed to JSON string. The JSON string will be sent to a JSP page by a http request to feed into heatmap.

Expression data source FlyExpressionScoreConverter is a specific data converter for modENCODE fly expression data. The class is located at *bio/sources/modmine/fly-expression-score*. Any other similar expression data conversion tasks can take the data source as a reference.

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

```
# modmine/dbmodel/resources/modencode-metadata_additions.xml
<class name="GeneExpressionScore" is-interface="true">
    <attribute name="score" type="java.lang.Double" />
    <reference name="gene" referenced-type="Gene" reverse-reference="expressionScores" />
    <reference name="cellLine" referenced-type="CellLine" />
    <reference name="developmentalStage" referenced-type="DevelopmentalStage" />
    <reference name="submission" referenced-type="Submission" />
    <reference name="organism" referenced-type="Organism" />
</class>
```

A better practice would be to add the model extension to a source specific additions.xml under a source directory.

Controller The controller class HeatMapController is a component of Struts MVC framework. It holds the logic to process user requests, and seletcs a proper wabpage to user.

In HeatMapController, a query is run to fetch expression scores from database (ref method *queryExpressionScore*), then the results are parsed to JSON string (ref method *getJSONString*) and set in the request (ref method *findExpression*).

Struts config:

```
# modmine/webapp/resources/struts-config-model.xml
<action path="/initHeatMap"
        type="org.modmine.web.HeatMapController" />

<action path="/heatMap" forward="heatMap.page" />


#  modmine/webapp/resources/tiles-defs-model.xml
<definition name="heatMap.tile" path="/model/heatMap.jsp"
        controllerUrl="/initHeatMap.do"/>

<definition name="heatMap.page" extends="layout.template">
        <put name="body" value="heatMap.tile"/>
        <put name="pageName" value="heatMap"/>
</definition>
```

Web page heatMap.jsp displays heatmap. canvasXpress object takes expression JSON string and other parameters in to create a heatmap (in modMine, we have two separate heatmaps for cell line and developmental stage respectively). jQuery was used to adjust page layout.

Further development

A modern way of creating widget like heatmap would be using InterMine webservices framework to query and generate JSON strings and embed heatmap on any web page. To learn more...

An alternative library would be D3.js, an example of heatmap can be found here. However canvasXpress is particular designed to display genomics data, D3 is for a broader use.

1.6.17 Monitoring Site Usage

Google Analytics

To enable Google Analytics to track usage of your webapp you need to set up your Analytics account and get a ‘‘code’‘ from Google then add a property to your *.intermine/xxxmine.properties* file:

```
google.analytics.id=CODE
```

This places the Google javascript to track usage views to every page of the webapp.

To modify the message that is going to be displayed to the user asking if they agree to the usage of cookies modify:

```
google.analytics.message=I accept cookies from this site
```

If no key - message is provided, no message is shown and cookies are accepted by default.

InterMine User Tracking

The following user activities are recorded during browsing of all mine webapps, to track the usage:

- template query executions
- query executions from the QueryBuilder
- list creations by upload, operations (copy, union...) or from result tables
- list executions
- searches by keyword
- user logins

The trackers, recording all tracks, are defined in the *global.web.properties* file under the property *webapp.trackers*.

All tracks are saved into specific tables, created automatically, if they don’t already exist, in the userprofile database.

The tables are:

- templatetrack
- querytrack
- listtrack
- searchtrack
- logintrack

The table are created at the startup of the webapp.

If some table definitions needed to be updated, the browsing of the webapp is blocked, and a warning message is displayed.

Run the ant task *update-templatetrack-table* in the webapp directory and then restart the webapp.

Usage page in the webapp

You can access to the *Usage* page, via the MyMine page, only if you are a superuser.

The page shows all tracks grouped in these sections: Template usage, Custom queries executed, Logins, Keyword Searches, List Analysis page views and List Creation.

Each section contains:

- a diagram showing the trend of that specific track during the time range selected: 1d, 5d, 1m, 3m, 6m, 1y. The diagrams point out the number of tracks per day (number of template query execution, number the query execution....) without specify wich template or type query(Gene, Protein...) has been executed.
- a table showing the number of tracks for each template, type of query, type of list....

Furthemore, in the ''Template usage" section, there is a pie diagram showing the first 9 most popular templates and their number of executions in the time range selected: last 2 weeks, last month, last 3 months, last year.

Under the label 'Other' the number of executions of templates from 10th to 15th position.

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="18FB9
...
</templatetracks>
<logintracks>
<logintrack username="butano@flymine.org" timestamp="2011-05-17 16:27:38.729"></logintrack>
...
</logintracks>
<listtracks>
<listtrack type="Company" count="2" buildmode="IDENTIFIERS" event="CREATION" username="butano@flymine
timestamp="2011-05-18 09:42:16.905"></listtrack>
...
</listtracks>
<querytracks>
<querytrack type="Address" username="butano@flymine.org" sessionidentifier="EDD0090DE148413B1B35E8DF
...
</querytracks>
<searchtracks>
<searchtrack keyword="CompanyA" username="" sessionidentifier="C83EAE50F49A7777E15C69AF31412839" time
...
</searchtracks></tracks></userprofiles>
```

Then run *create-db-userprofile* and *read-userprofile-xml*

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://siteexplorer.search.yahoo.com
searchengines.yahoo=CODE

# http://www.bing.com/webmaster
searchengines.msn=CODE
```

See also:

Google Analytics

"robots.txt"

The easiest way to control what the search engines index is to use a file called robots.txt. Robots use this file to determine which parts of the site they can visit. This file should be located in the root of your site, ie. www.flymine.org/robots.txt

You can also specify which search engines can index your site, e.g. Google or Yahoo. Here is an example file:

```
Sitemap: sitemap_index.xml

User-agent: *
Disallow: /

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

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

User-agent: msnbot
Disallow:
Disallow: /release-8.2/
Disallow: /release-8.1/
```

This file bans all search engine robots except for Google, Yahoo, and MSN. In addition this file forbids the robots to index files in the release-8.1 and release-8.2 directories.

Read more about this document on the http://www.robotstxt.org website.

NOFOLLOW

You can restrict access to directories via the robots.txt file, but you can also configure your site to allow or forbid access to specific web pages.

To prevent the search engine robots from following links on that page, set the noFollow attribute in the InterMineWebApp.properties file:

```
# MYMINE
mymine.title = MyMine
mymine.description = Your list of saved lists and queries
mymine.tab = mymine
mymine.noFollow = true
```

Sitemaps

Search engines often have difficulty indexing dynamic websites. The easiest solution for this is provide a sitemap that indicates which pages should be indexed.

1.6.18 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: *bio/tutorial/malariamine/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.

Property name	Example	Determines
db.production.datasource.serverName	server_name	server name
db.production.datasource.databaseName	malariamine	database name
db.production.datasource.user	postgres_user	database username
db.production.datasource.password	SECRET	database password

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

Property name	Example	Determines
db.common-tgt-items.datasource.serverName	server_name	server name
db.common-tgt-items.datasource.databaseName	malariamine	database name
db.common-tgt-items.datasource.user	postgres_user	database username
db.common-tgt-items.datasource.password	SECRET	database password

The following properties determine the settings for the items database. This database is used by the webapp only. It holds all user related information, including lists, queries and tags.

Property name	Example	Determines
db.userprofile-production.datasource.serverName	server_name	server name
db.userprofile-production.datasource.databaseName	malariamine	database name
db.userprofile-production.datasource.user	postgres_user	database username
db.userprofile-production.datasource.password	SECRET	database password

Web application name and location

Property name	Example	Determines
os.production.verboseQueryLog	true	if true, all queries are logged. Defaults to false
webapp.deploy.url	http://localhost:8080	location of tomcat server
webapp.path	malariamine	location of path of webapp
webapp.manager	TOMCAT_USER	tomcat username, needed to deploy webapp
webapp.password	TOMCAT_PWD	tomcat password, needed to deploy webapp
webapp.baseurl	http://www.flymine.org	home link; used by client side JavaScript AJAX requests
superuser.account	test_user@mail_account	account name for superuser
superuser.initialPassword	secret	password used when account is created
project.standalone	true	run with associated web site. Defaults to false
project.title	MalariaMine	name of mine
project.subTitle	An example of InterMine.bio with data from Plasmodium falciparum	text that appears in the header at the top of the page
project.releaseVersion	tutorial	text that appears at the top of the page next to the mine name
project.sitePrefix	http://www.flymine.org	various URLs use this as the prefix
project.helpLocation	http://www.flymine.org/help	various URLs use this as the prefix

Warning: *webapp.baseurl* and *webapp.path* must be correct or else your queries will not run

Email

Emails are sent to users when they create an account, forget their password, or use the contact form.

Property name	Example	Determines
mail.host	localhost	mail host to use
mail.from	account@my_mail_host	"from" email address
mail.subject	Welcome to MalariaMine	"subject" for email send when account created
mail.text	You have successfully created an account on MalariaMine	"body" for email send when account created
feedback.destination	test_user@mail_address	recipient of feedback form located on bottom of every page

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 Link 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

portal.welcome the message to show when a user arrives at the webapp via the portal action (eg. <something>/portal.do)

quickSearch.identifiers Text displayed in search box

theme Colour scheme for the webapp. Available options are: blue, bright_blue, gold, green, grey, brown, ecoli_blue, metabolic, modmine, ratmine and purple

xrefLink Used to configure hyperlinks for CrossReferences. See below

Home page

Search box (first box on the left)

begin.searchBox.title	title of box on left
begin.searchBox.description	text in the box on the left
begin.searchBox.example	text in the form field
textarea.identifiers	text present on list upload on home page

List upload box (middle box)

begin.listBox.title	Title of box
begin.listBox.description	Text in box
begin.listBox.example	Text in form field

Third box

begin.thirdBox.title	Title of box if user is new
begin.thirdBox.visitedTitle	Title of box if user has visited before
begin.thirdBox.description	Text in box
begin.thirdBox.linkTitle	Text for large button
begin.thirdBox.link	URL for large button

Tabs

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

begin.tabs.1.id	Name of category, eg. Genes
begin.tabs.1.description	Text for that tab

List upload examples

Using the *bag.example.identifiers* key, one can provide a list of keyword examples on the list create/upload page. This could lead to a mixed list of items being updated and only, say Protein or Gene, identifiers being uploaded.

If one wants to provide different example identifiers per different types, like Genes, Proteins, SNPs, you can do so by using the following key:

bag.example.identifiers.gene, *bag.example.identifiers.protein* etc.

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

list indicates the link will have a list of identifiers

url url to link to

imageName name of logo (optional), must be in /model directory

text text that will appear next to the logo

The value of the attribute (for the current object) is substituted anywhere the string "<<attributeValue>>" occurs in the text or the url

example:

```
attributelink.flybase.Gene.7227.primaryIdentifier.url=http://www.flybase.org/.bin/fbidq.html?<<attrib
attributelink.flybase.Gene.7227.primaryIdentifier.text=FlyBase: <<attributeValue>>
```

In this case *Gene* pages for Drosophila melanogaster will have a link that uses the *organismDbId* field.

A list example:

```
attributelink.flymine.Gene.*.primaryIdentifier.list.url=http://www.flymine.org/query/portal.do?extern
attributelink.flymine.Gene.*.primaryIdentifier.list.text=FlyMine
attributelink.flymine.Gene.*.primaryIdentifier.list.imageName=flymine_logo_link.gif
```

Only if a taxonId is specified the code will check if the link to the external db is relevant.

Settings for the xrefLink property

You can configure the URLs for querying CrossReference from external sources by adding entries to the {{{web.properties}}} file.

The format for this property is:

```
# on the report page
'xreflink' + dataSource_name + (url|imageName)
```

dataSource_name the name of the external database

url url to link to

imageName name of logo (optional), must be in /model directory

example:

```
xreflink.PFAM.url=http://pfam.sanger.ac.uk/family?
xreflink.PIRSF.url=http://pir.georgetown.edu/cgi-bin/ipcSF?id=
```

Cross references represent identifiers used in external databases, eg. FlyBase, UniProt. An object in InterMine which has CrossReference will have a identifier and data source for that cross reference. In order to find the cross reference in that data source, a url is required to link to and the full path should look like url+identifier, e.g. ''http://pfam.sanger.ac.uk/family?PF00001''. In web.properties, the first part of the full path could be configured as in "url", and identifier will be added programmatically to the rear of it. The dataSource_name should be consistent with the source name of the CrossReferences in the InterMine database.

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.

Data and Widget Configuration

The *webconfig-model.xml* file configures aspects of how data appears on the InterMine webapp.

This file allows for inheritance - a subclass will inherit from its parent class but only if that subclass has no configuration. Configuration settings for the parent class do not overwrite settings for the subclass.

Field Configuration

You can configure which fields are displayed on report and result pages for each class in your model.

attribute name	purpose	required?	default
fieldExpr	field name	yes	•
label	human readable name	no	generated automagically
showInInlineCollection	show field in inline collection (on report pages)	no	true
showInSummary	add field to query when user clicks on 'Summary' button in QueryBuilder	no	true
showInResults	show field in results table	no	true
doNotTruncate	don't truncate display	no	false
fieldExporter	specify class to export file field	no	•
sectionOnRight	show on the right side of the page	no	false
sectionTitle	if sectionOnRight="true", title for section on right	no	•
openByDefault	if sectionOnRight="true", whether or not this section should be open	no	false

For example:

```
<class className="org.flymine.model.genomic.Protein">
  <fields>
    <fieldconfig fieldExpr="primaryIdentifier"/>
    <fieldconfig fieldExpr="primaryAccession"/>
    <fieldconfig fieldExpr="organism.name"/>
    <fieldconfig fieldExpr="length" displayer="/model/sequenceShortDisplayerWithField.jsp" />
  </fields>
  <bagdisplayers>
   < -- attribute links can now be displayed on protein list analysis pages -->
   <displayer src="attributeLinkDisplayer.tile"/>
  </bagdisplayers>
</class>
```

Displaying Data on Report pages

ReportDisplayers allow custom display of particular data types on report pages, typically to replace default tables with more appropriate presentation of data.

```
<reportdisplayer javaClass="org.intermine.bio.web.displayer.CytoscapeNetworkDisplayer"
                 jspName="model/cytoscapeNetworkDisplayer.jsp"
                 replacesFields="interactions"
                 placement="Interactions"
```

Export Configuration

Users can export data from InterMine in comma or tab-delimited files. InterMine also allows for the addition of custom exporters. To add a custom exporter, create a Java class to format the data and add an entry to the web config file, for example:

```
<tableExportConfig id="sequenceExporter" actionPath="/exportAction?type=sequenceExporter"
                   className="org.intermine.bio.web.export.SequenceHttpExporter"/>
<tableExportConfig id="gff3Exporter" actionPath="/exportAction?type=gff3Exporter"
                   className="org.intermine.bio.web.export.GFF3HttpExporter"/>
```

Widget Configuration

At the bottom of the config file are the configuration entries for widgets. Please see [wiki:Widgets] for detailed information about how to configure widgets.

```
<enrichmentwidgetdisplayer id="publication_enrichment"
                           title="Publication Enrichment"
                           description="Publications enriched for genes in this list."
                           label="Publication"
                           startClass="Gene"
                           startClassDisplay="primaryIdentifier"
                           enrich="publications.title"
                           enrichIdentifier="publications.pubMedId"
                           constraints="organism.name=[list],primaryIdentifier  = null"
                           typeClass="org.intermine.model.bio.Gene"
                           views="secondaryIdentifier, symbol, organism.name,
                                 publications.title, publications.firstAuthor,
                                 publications.journal, publications.year, publications.pubMedId"
                           externalLink="http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&a
```

1.6.19 Data Categories

The features this page discusses are deprecated. Their use is not recommended in new mines

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

```
<aspect name="Genomics">
  <subtitle>Genome annotation</subtitle>
  <icon-image>model/genomics.gif</icon-image>
  <large-image>model/genomics.gif</large-image>
  <intro-text>
    The gene structure and other genome annotation in FlyMine are provided by
    a variety of curated source databases.  In most cases FlyMine includes
    all genome features that have chromosomal locations (eg. genes and repeat regions).
```

```
  </intro-text>
  <tile-name>model/genomics.jsp</tile-name>
  <aspect-source name="FlyBase" url="http://www.flybase.org"/>
  <aspect-source name="Ensembl" url="http://www.ensembl.org/Anopheles_gambiae"/>
</aspect>
```

Configuration

- logo
  - <icon-image>model/genomics.gif</icon-image> - appears on the home and data category pages
  - <large-image>model/genomics.gif</large-image> - appears on the individual data category page
- short description
  - <intro-text>TEXT HERE</intro-text>
  - appears on the top of the data category page
- external links
  - <aspect-source name="FlyBase" url="http://www.flybase.org"/>
  - appear on the top right corner of the data category page
- bulk download queries
  - appear on the top right corner of the data category page
- template queries
  - appear on the data category page
- direct links to QueryBuilder
  - links will appear at the bottom of the data categories page

To configure which template queries appear on a data category page, tag the template.

Data page/tab

The data tab points to this JSP file *intermine/webapp/main/resources/webapp/dataCategories.jsp*. You can overwrite this file and display your own customised file by putting a JSP in your */webapp* directory.

1.6.20 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.

1.7 Web Services

InterMine allows, and encourages, programmatic access to its features, also known as *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 please see the HTTP API documentation.

1.7.1 Getting Started

Install Required Dependenices 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.

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).

For the kinds of things you will probably be wanting to do, see howtows

1.7.2 API and Client Libraries

InterMine exposes its functionality over an HTTP API (generally following RESTful principles, but there is a bit of RPC there). Client libraries are available in commonly used languages so you can get started quickly. All our code is hosted on Github, and contributions are welcome. All InterMine code is free and open-source, released under the LGPL (see *Legal*).

For information on the underlying API, and the supported libraries, please visit the following links:

HTTP API Documentation: http://iodocs.labs.intermine.org

Java Download | Docs | Source

Perl Client Download | Docs | Source

Python Client Download | Docs | Source

Ruby Client Download | Docs | Source

JavaScript Client (for Browser and node.js) Download | Docs | Source

1.8 Embedding InterMine components

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

1.8.1 InterMine JavaScript API Loader

See also:

GitHub repo for source code. For a live example refer to FlyMine list widgets on Tinkerbin.

Purpose

1. To simplify delivery of InterMine's JavaScript clients.
2. JS Api provides a common namespace *intermine* for all our JavaScript.
3. Can/will be utilized as a loader of dependencies such as jQuery, underscore etc. whenever needed.
4. As JavaScript is namespaced, we can determine which of our libraries are already loaded.

How to use

In an embedding context, the only dependency would be the jsapi, hosted remotely. We provide a callback and wait for the API to load the Widgets library.

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

```
intermine.load('widgets', function() {
    var widgets = new intermine.widgets('http://flymine.org/service');
});
```

In a mine context we want to serve resources locally. As we include jQuery locally this is recognized by jsapi and no internet connection is required.

```
// include jQuery locally
<script src="js/jquery.js"></script>

// point to API, requirement for all InterMine client side JavaScript
<script src="js/intermine.api.js"></script>
// include Widgets library locally, is immediately available on the `intermine` namespace
<script src="js/intermine.widgets.js"></script>
```

```
var widgets = new intermine.widgets('http://flymine.org/service');
```

Asking for a specific version of a library to be loaded.

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

```
intermine.load('widgets', '0.9.1', function() {
    var widgets = new intermine.widgets('http://flymine.org/service');
});
```

Within a JavaScript library, load resources as asynchronously as possible.

```
var resources = [
    {
        name: "jQuery",
        path: "http://cdnjs.cloudflare.com/ajax/libs/jquery/1.7.2/jquery.min.js",
        type: "js",
        wait: true
    }
];
```

```
intermine.load(resources, function() {
    // ...
});
```

1.8.2 InterMine JavaScript Library

Please refer to this repo for more information.

1.8.3 Embedding examples

Please refer to this repo for various embedding examples using List Widgets, imjs and the like.

See also:

Report Displayers if you wish to embed a displayer on a report page only.

1.9 Support

1.9.1 Mailing list

Please join the InterMine developers mailing list `dev [at] intermine [dot] org` to receive updates and ask questions.

Join list

Archives of old messages are available here:

Message archive

1.9.2 Troubleshooting tips

This page describes what to do if you encounter problems when installing or maintaining InterMine. Please feel free to [wiki:ContactUs contact us] with any questions you may have.

Error messages

If you encounter a problem when running Ant, try adding the verbose flag:

```
$ 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:

```
$ tail -f tomcat/logs/LOGNAME
```

If you reload the webapp you will see the error output directly on the screen.

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
```

This means that your version of Java is too old, you need at least Java 1.5 to run !InterMine.

can't open datasource

```
java.lang.RuntimeException: can't open datasource for {platform=PostgreSQL, datasource.dataSourceName
```

Errors of this type mean there was a problem accessing a database, in this example with *db.flatmodeunittest*. Either the database specified in the 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 -
```

This occurs when the number of connections to a database exceeds the maximum configured in the postgres configuration. You need to increase the value of *max_connections* in the *postgresql.conf* file and restart postgres. Try 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.inter
```

In this example a class named *Protein* could not be found in the data model, this will usually arise when running a parser and attempting to create an *Item* for a class that does not exist. Check your *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*?

Can't find antlib-int.xml

```
Unable to find jar:file:/home/malariamine/svn/trunk/intermine/integrate/main/dist/intermine-integrate
```

This happens occasionally, probably due to a concurrency issue in the build system. Run the ANT command again and it should work.

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);
```

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.

1.9.3 Contact us

To contact the InterMine Team:

email `info [at] intermine [dot] org`

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 known to expect you) We are building number 9 on this map. Please note that the front door to this building is only accessible from Tennis Court Road. Street view.

1.9.4 Upgrade Instructions

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

```
$ git pull upstream
```

If you don't have a git repo yet, see Get the Software for details.

Upgrade to InterMine 1.1

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

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

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

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

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

Model Changes

Updated to latest version of Sequence Ontology, 2.5

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

Interactions

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

Gene Ontology

class	old	new
GOAnnotation	withText	evidence.withText
	with	evidence.with
	–	annotationExtension
OntologyTerm	–	crossReferences [1]

Identifiers

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

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

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

Configuration Updates

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

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

Tomcat 7.0

If you would like to use Tomcat 7.0 with InterMine 1.1, add this property:

```
# in ~/.intermine/MINE_NAME.properties
webapp.tomcat.version=7
```

[1] used for Uberon

1.10 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 rat genomic data, developed by SGD
- RatMine - an integrated data warehouse of yeast 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:

1.10.1 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 modMine database contains 130 million objects, and its size with precomputed tables is 750 GB, with metabolicMine being even larger, containing 260 million objects and almost 2000GB including precomputed tables.

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.

1.10.2 How to cite us

If you use the InterMine framework in your research, we would appreciate it if you cite the following publication:

- InterMine: 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:

- FlyMine: an integrated database for Drosophila and Anopheles genomics. Lyne, R. et al. Genome Biol. 8, R129 (2007).
- MitoMiner: a data warehouse for mitochondrial proteomics data. Smith, A. C., Blackshaw, J. A. & Robinson, A. J. Nucleic Acids Res. 40, D1160-1167 (2012).
- modMine: flexible access to modENCODE data. Contrino, S. et al. Nucleic Acids Res. 40, D1082-1088 (2012).
- TargetMine, an integrated data warehouse for candidate gene prioritisation and target discovery. Chen, Y. A., Tripathi, L. P. & Mizuguchi, K. PLoS One 6, e17844 (2011).
- YeastMine - an integrated data warehouse for Saccharomyces cerevisiae data as a multipurpose tool-kit. Balakrishnan, R. et al. Database 2012, bar062 (2012).

1.10.3 Legal

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

1.10.4 Privacy Policy

Privacy policy

This privacy policy sets out how InterMine uses and protects any information that you give when you use InterMine websites.

InterMine is committed to ensuring that your privacy is protected. Should we ask you to provide certain information by which you can be identified when using InterMine websites, then you can be assured that it will only be used in accordance with this privacy statement.

What we collect

Tracking We log the IP address of your browser to track usage statistics and to identify operational problems. This information is not used to identify individuals or organizations, and is never shared with third parties.

Cookies are used to provide persistence across browsing sessions. These may persist after you exit your browser, but they are never used for either identification or tracking purposes.

Login Details If you choose to create an account to save your data, we save your username and password information. This information is not used to identify individuals or organizations, and is never shared with third parties.

What we do with the information we gather

We require this information to understand your needs and provide you with a better service, and in particular for the following reasons:

- Tracking **usage statistics**.
- Identifying **operational problems**.
- Allowing you to **log in** and save your data.

The information we collect is not used to identify individuals or organizations, and is never shared with third parties.

Security

We are committed to ensuring that your information is secure. In order to prevent unauthorised access or disclosure, we have put in place suitable procedures to safeguard and secure the information we collect online.

How we use cookies

A cookie is a small file which asks permission to be placed on your computer's hard drive. Once you agree, the file is added and the cookie helps analyse web traffic or lets you know when you visit a particular site. Cookies allow web applications to respond to you as an individual. The web application can tailor its operations to your needs, likes and dislikes by gathering and remembering information about your preferences.

We use traffic log cookies to identify which pages are being used. This helps us analyse usage data and identify operational problems. We only use this information for statistical analysis purposes and then the data is removed from the system. Overall, cookies help us provide you with a better website, by enabling us to monitor which pages you find useful and which you do not, and saving your lists and queries across browsing sessions. A cookie in no way gives us access to your computer or any information about you, other than the data you choose to share with us.

You can choose to accept or decline cookies. Most web browsers automatically accept cookies, but you can usually modify your browser setting to decline cookies if you prefer. This may prevent you from taking full advantage of the website.

Links to other websites

Our website may contain links to other websites of interest. However, once you have used these links to leave our site, you should note that we do not have any control over that other website. Therefore, we cannot be responsible for the protection and privacy of any information which you provide whilst visiting such sites and such sites are not governed by this privacy statement. You should exercise caution and look at the privacy statement applicable to the website in question.

Controlling your personal information

We will not sell, distribute or lease your personal information to third parties. We will only use your personal information to send you information if you specifically sign up to our mailing lists, and you can opt out of receiving these at any time.

1.10.5 Release Notes

Release Notes for InterMine 1.1

In this release, we've added lots of new features - users can share lists, mines can have more than one "superuser" and users can change the background distribution of enrichment widgets. We've also improved the model for interactions and gene ontology.

Note: This release is only available via GitHub.

See Upgrade Instructions for instructions on how to update your Mine to the new version.

Report Widgets

You can now embed widgets in any HTML page. See http://reportwidgets-intermine.rhcloud.com for live, interactive examples.

See Report Widgets for details on how to use these for your mine.

List Widgets

Users can now change and save the background distribution for enrichment widgets on list analysis page.

Multiple Admins

The main superuser can "deputise" other users to give them superuser powers - meaning they can publish public templates and lists in the mine.

Sharing Lists

Users can share lists with registered users or invite new users to share lists.

Data

We've added a several new data sources

- Uberon
- identifiers
  - flybase-identifiers
  - zfin-identifiers
  - wormbase-identifiers
  - sgd-identifiers
- Homologues
  - OrthoDB
  - Panther
  - Homologene

Export

When exporting sequences you can now specify the size of flanking regions to include.

Model Changes

- GO
- interactions

Please see Upgrade Instructions for the complete list of changes to the model.

Other

- UPDATE - list upgrade performance significantly improved.
- FIX - if a friendly mine is unavailable, this no longer causes a delay in pages being loaded
- FIX - inner joins now work in all queries
- FIX - lots of minor bug fixes, see the complete list https://github.com/intermine/intermine/issues?milestone=2
- NEW - InterMine works with Tomcat 7.0

CHAPTER 2

Indices

- genindex
- site-map
- search

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

W

X

Y

Z