

Xref system

- Parsing the external database references
- Mapping the external database references to the Ensembl core database
 - 1) process the config file
 - 2) dump the fasta files
 - 3) load core data into the xref database
 - 4) run exonerate to produce the mapping files
 - 5) process the mapping files
 - 6) process the direct xrefs
 - 7) flag the best priority xrefs
 - 8) process paired data
 - 9) check if any source is on more than one ensembl object type and fix
 - 10) official naming (for human, mouse and zebrafish in release 68)
 - 11) checks
 - 12) coordinate mapping
 - 13) load data into the core
 - 14) populate display xref for genes and transcripts
 - 15) **populate gene descriptions**

The process of mapping external references to Ensembl entities consists of two parts. First part is **parsing the data** into a temporary database (Xref database).

The second part is to **map the new Xrefs to Ensembl genes, transcripts and translations**.

Parsing the external database references

In the xref_mapper directory you will find an ini-file called 'xref_config.ini'. This file contains two types of configuration sections:

- source and
- species.

A source section defines Xref priority, order and also the URIs pointing to the data files that the source should use (as key-value pairs, see the comment at the top of the source sections for a fuller explanation of these keys).

A species section contains information about species aliases, the numerical taxonomy ID(s) and what sources to use for that species.

If a species has more than one taxonomy ID (in the case where there are multiple strains or subspecies, for example), there can be more than one 'taxonomy_id' key.

Script 'xref_config2sql.pl' is used to convert the ini-file into an SQL file which generates 'sql/populate_metadata.sql'.

The 'xref_config2sql.pl' script expects to find 'xref_config.ini' in the current directory, but you may specify an alternative file as the first command line argument to the script if you have moved or renamed the ini-file. When 'xref_parser.pl' is run it will load the generated SQL file into the database

and will then download and parse all external data files for one or several specified species.

The parsing can create three types of Xrefs these are

1. Primary (These have sequence and are mapped via exonerate)
2. Dependent (Have no sequence but are dependent on the Primary ones)
3. Direct (These are directly linked to the Ensembl entities, so the mapping is already done)

Some sources will have more than one set of files associated with it, in these cases they have the same source name but different source IDs.

These are known as "priority Xrefs" as the Xrefs are mapped according to the priority of the source. An example of this is HGNCs.

Mapping the external database references to the Ensembl core database

1) process the config file

Get information on the databases to do the mapping on. Also store the options passed to the program and info on the databases to the meta table.

meta_id	species_id	meta_key	meta_value	date
5	1	xref	host1:macaca_xref	2009-05-26 16:56:52
6	1	species	host2:macaca_core	2009-05-26 16:56:52
7	1	mapper options	-file xref_input	2009-05-26 16:56:52

2) dump the fasta files

Primary Xrefs are dumped out to Fasta files, Ensembl Transcripts and Translations are then dumped out to two files in Fasta format.

If -dumpcheck is specified then the system checks to see if the fasta files exist already and if it does, does not redump the fasta files.

Dumping the fasta files from the core database can be one of the longest steps so if the core fasta files exist already this is very useful.

The fasta file will be written to where ever was specified in the configuration file.

The process_status table should now read something like :

id	status	date
1	xref_created	2009-05-26 16:21:43
2	parsing_started	2009-05-26 16:21:43
3	parsing_finished	2009-05-26 16:54:25
4	xref_fasta_dumped	2009-05-26 16:57:08
5	core_fasta_dumped	2009-05-26 16:57:08

3) load core data into the xref database

For ease of use and to reorganise some data we copy data from the core database to the xref database.

gene_stable_id, transcript_stable_id and translation_stable_id are all copied as is from the core database.

gene_transcript_translation table stores information regarding as to how these are all linked in one easy table.

Information about the external databases that are "KNOWN" or "KNOWNXREF" are stored for that source in the source table,

as this is needed for the gene/transcript status calculation later on.

4) run exonerate to produce the mapping files

Exonerate is used to find the best matches for the Xrefs.

If there is more than one best match then the Xref is mapped to more than one Ensembl entity.

A cutoff is used to filter the best matches to make sure they pass certain criteria. By default this is that the query identity

OR the target identity must be over 90%. This can be changed by creating your own '<method>.pm' file in the directory

'XrefMapper/Methods' and creating subroutines 'query_identity_threshold()' and 'target_identity_threshold()' which

return the new values.

Additionally, for RefSeq sources mappings between RefSeq xrefs and Ensembl transcripts must match on biotype

(e.g. RefSeq_ncRNA can't map to a non-protein-coding transcript).

So exonerate will generate a set of .map files with the mapping in. The map-files are then parsed and any that pass the criteria

are stored in the 'xref' table, 'object_xref' table and the 'identity_xref' table. All dependent Xrefs are also stored if the parent is mapped.

Any Xrefs which fail to be mapped are written to the unmapped_object table with a brief explanation of why they could not be mapped.

The mapper uses exonerate to produce the mapping files. If the option -nofarm is used then exonerate will run locally.

The mapper sets the exonerate jobs running and then writes to the tables mapping_jobs and mapping storing information about these jobs.

The table mapping holds information about what mapping method and the mapping_jobs table holds information about the individual exonerate jobs. For most runs there will be 2 entries in the mapping table one for the alignment of the dna and another for the peptides:

```
[user@machine] [macaca_xref]> select * from mapping \G
***** 1. row *****
      job_id: 76917
      type: dna
      command_line: exonerate-1.4.0 xref_0_dna.fasta macaca_mulatta_dna.fasta
                   --querychunkid $LSB_JOBINDEX --querychunktotal 586
                   --showvulgar false --showalignment FALSE
                   --ryo "xref%qi:%ti:%ei:%ql:%tl:%qa:%qae:%tab:%tae:%C:%s\n"
                   --gappedextension FALSE --model affine:local --subopt no
                   --bestn 1
                   | grep "^xref" > ExonerateGappedBest1_dna_$LSB_JOBINDEX.map
      percent_query_cutoff: 90
      percent_target_cutoff: 90
      method: ExonerateGappedBest1
      array_size: 586
***** 2. row *****
      job_id: 76919
      type: peptide
      command_line: exonerate-1.4.0 xref_0_peptide.fasta
                   macaca_mulatta_protein.fasta
                   --querychunkid $LSB_JOBINDEX --querychunktotal 73
                   --showvulgar false --showalignment FALSE
                   --ryo "xref:%qi:%ti:%ei:%ql:%tl:%qa:%qae:%tab:%tae:%C:%s\n"
                   --gappedextension FALSE --model affine:local --subopt no
                   --bestn 1
                   | grep "^xref" > ExonerateGappedBest1_peptide_$LSB_JOBINDEX.map
      percent_query_cutoff: 90
      percent_target_cutoff: 90
      method: ExonerateGappedBest1
      array_size: 73
```

So here we can see that the dna alignment was split into 586 separate farm jobs. The percentage cutoffs are used when the mapping files are processed and not by exonerate itself.

We can see the individual jobs ran by looking at the table mapping_jobs:-

```
[user@machine] [macaca_xref]>select map_file, status, array_number, job_id from
mapping_jobs limit 2;
+-----+-----+-----+-----+
| map_file           | status   | array_number | job_id |
+-----+-----+-----+-----+
| ExonerateGappedBest1_dna_1.map | SUBMITTED | 1 | 76917 |
| ExonerateGappedBest1_dna_2.map | SUBMITTED | 2 | 76917 |
+-----+-----+-----+-----+
```

So here we can see the map file that will be produced and the array number for a particular job_id.

After the exonerate jobs have been issued to the farm a depend job is set to wait for all the exonerate jobs to finish.

If the farm is not used then since the jobs are run locally no depend job is needed.

5) process the mapping files

Several checks are made while processing the mapping files and if the farm was used then checks are made on its error files to make sure they are empty.If any errors are found then the status for this mapping job is set to "FAILED"

and the program exits gracefully after all the files have been read. So if there is an error in one of the mapping files

then this is noted but all the other mapping files are still read in afterwards but the program then exits.

When the first entry is read from a mapping file the first object_xref that is stored has its id stored in the column object_xref_start in the table object_xref_start and also the last object_xref that is stored has its id stored into the object_xref_end column.

So from this we know for each mapping file the range of object_xrefs that have been stored. If the mapping file is processed with no errors then the status is set to "SUCCESS".

```
select map_file, status, array_number as arr, job_id, object_xref_start as start,
object_xref_end as end from mapping_jobs limit 2;
```

map_file	status	arr	job_id	start	end
ExonerateGappedBest1_dna_1.map	SUCCESS	1	76917	1	461
ExonerateGappedBest1_dna_2.map	SUCCESS	2	76917	462	674

The processing of the mapping file creates entries in the object_xref and identity_xref tables for the primary xrefs and also its dependents.

Also entries may be added to the go_xref table if there are GO xrefs. In the object_xref table we set the default ox_status to be "DUMP_OUT" if the

mapping passes the percentage cutoff criteria else it is set to "FAILED_CUTOFF", so that we know which mapping are good.

6) process the direct xrefs

Direct xrefs are those xrefs where we have a direct mapping is taken from a file or database. The mapper is not used for these ones as the mapping is already specified. So we now take all the entries from the tables gene_direct_xref, transcript_direct_xref and translation_direct_xref and create the object_xrefs for these. If the stable_id cannot be found at this point a warning is given.

7) flag the best priority xrefs

For priority Xrefs (ones that have multiple sources) the highest priority one is stored, and the rest are discarded.

Priority xrefs are those xrefs where the external database xrefs may come from several different sources that have different priorities.

A good example here is the HGNC xrefs for human:

```
> select source_id as id, name, priority, priority_description from source where
name like "HGNC";
```

id	name	priority	priority_description
53	HGNC	1	havana
54	HGNC	2	ccds
55	HGNC	4	entrezgene_manual
56	HGNC	5	refseq_manual
57	HGNC	6	entrezgene_mapped
58	HGNC	7	refseq_mapped
59	HGNC	8	uniprot
60	HGNC	3	ensembl_mapped
61	HGNC	100	desc_only

So we can see that HGNC has 9 sources where havana (manual annotation) has the best priority.

What the flag priority xrefs step does is sets the ox_status in the object_xref table to be "FAILED_PRIORITY"

for those where there is a better match (better priority).

Here is an example:

```
>select x.label, x.info_type, s.name, s.priority, s.priority_description,
       ox.ox_status
       from xref x, source s, object_xref ox
       where x.source_id = s.source_id and ox.xref_id = x.xref_id
             and s.name like "HGNC" and label like "BRCA2" limit 10;
```

label	info_type	name	priority	priority_description	ox_status
BRCA2	DEPENDENT	HGNC	8	uniprot	FAILED_PRIORITY
BRCA2	DEPENDENT	HGNC	5	refseq_manual	FAILED_PRIORITY
BRCA2	DEPENDENT	HGNC	7	refseq_mapped	FAILED_PRIORITY
BRCA2	DEPENDENT	HGNC	4	entrezgene_manual	FAILED_PRIORITY
BRCA2	DEPENDENT	HGNC	6	entrezgene_mapped	FAILED_PRIORITY
BRCA2	DIRECT	HGNC	1	havana	DUMP_OUT
BRCA2	DIRECT	HGNC	2	ccds	FAILED_PRIORITY

So because the havana one is the best we set the ox_status for the other matches in the object_xref table to be "FAILED_PRIORITY" for BRCA2. From this point on the "FAILED_PRIORITY" object xrefs are ignored.

8) process paired data

This applies to RefSeq_peptide and RefSeq_mRNA which are considered pairs. If RefSeq_peptide xref was mapped to ensembl protein and it's corresponding RefSeq_mRNA was mapped to the transcript which produces the protein - this is treated as the best match.

Delete remaining RefSeq_peptide matches if this best match exists.

Also match the RefSeq_peptide xref to a translation if it's transcript was sequence matched to the corresponding RefSeq_mRNA.

9) check if any source is on more than one ensembl object type and fix

Because Biomart does not like having any particular data source on more than one ensembl type (gene/transcript/translation) this check searches for these instances and moves all the object_xrefs onto the highest level. So if HGNC is on Genes and Transcripts then all the one on Transcripts will be moved to the corresponding Genes.

10) official naming (for human, mouse and zebrafish in release 68)

Because we want to have the same names (with a postfix) for all the transcripts in a gene, this process gets the best name (HGNC/MGI) for a gene

taken from any of its transcripts and then applies this to all the transcripts and gene. An example here is PDS5B which has two transcripts PDS5B-006 and PDS5B-201. If the postfix starts with a 0 it means that this comes directly from havana.

If it starts with a 2 then the transcript is not found in havana.

11) checks

There is a list of checks which are performed. Some check primary/foreign key pairs, others check the number of xref and object_xrefs

in the xref database compared to the core database. Depending on the seriousness of the problem a warning is given and then may exit gracefully.

12) coordinate mapping

For human and mouse we map UCSC stable ids to our system using their locations.

13) load data into the core

First for each source that is in the xref database the corresponding data is deleted from the core database.

This includes xref, object_xref, identity_xref, external_synonym, go_xref, dependent_xref and unmapped_object tables.

Also all the Projected xrefs are deleted. Via some complex sql these tables are now filled with the new data.

14) populate display xref for genes and transcripts

The external databases to be used for the transcript and gene display_xrefs are taken from either the DisplayXrefs.pm

subroutines transcript_display_sources and gene_display_sources respectively, or [species name].pm if the methods were overloaded:

```
sub transcript_display_xref_sources {
    my $self = shift;

    my @list = qw(RFAM
                  miRBase
                  Uniprot/SWISSPROT
                  Uniprot/VarSplic
    );

    my %ignore;

    return [ \@list, \%ignore ];
}

sub gene_display_xref_sources {
    my $self = shift;

    my @list = qw(RFAM
                  miRBase
                  Uniprot_genename
                  EntrezGene);

    my %ignore;

    #don't use EntrezGene labels dependent on predicted RefSeqs
    $ignore{'EntrezGene'} =<<IEG;
    SELECT DISTINCT ox.object_xref_id
    FROM object_xref ox, dependent_xref dx,
         xref xmas, xref xdep,
         source smas, source sdep
    WHERE ox.xref_id = dx.dependent_xref_id AND
          dx.dependent_xref_id = xdep.xref_id AND
          dx.master_xref_id = xmas.xref_id AND
          xmas.source_id = smas.source_id AND
          xdep.source_id = sdep.source_id AND
          smas.name like "Refseq%predicted" AND
          sdep.name like "EntrezGene" AND
          ox.ox_status = "DUMP_OUT"
    IEG

    #don't use labels starting with LOC
    $ignore{'LOC_prefix'} =<<LOCP;
    SELECT object_xref_id
    FROM object_xref JOIN xref USING(xref_id) JOIN source USING(source_id)
    WHERE ox_status = 'DUMP_OUT' AND label REGEXP '^LOC[:digit:]*'
    LOCP

    return [ \@list, \%ignore ];
}
```

15) populate gene descriptions

As above, we use the sub gene_description_sources.

