

# The Ensembl Xref System

## Overview

This document describes the new (November 2004) Ensembl xref system.

The xref process consists of two steps:

1. Downloading xref data, parsing it and loading it into the xref holding database
2. Mapping the xrefs in the holding database to an Ensembl object (gene, transcript, or translation)

The two stages are separate and can be run at different times, although obviously xrefs that haven't been parsed can't be mapped.

## Terminology

xref – an external reference, i.e. a link to something that is stored in a database outside of Ensembl.

source – one of the databases or resources which we produce links to.

primary xref – an xref that has associated sequence, e.g. SwissProt.

dependent xref – an xref that is a “child” of another (usually primary) xref, e.g. PubMed xrefs are considered dependent on the SwissProt xref in which they are referenced.

synonym – an alternative name for an xref.

direct xref – one where the mapping has already been done outside Ensembl (e.g. CCDS)

## The xref holding database

When the xrefs are parsed they are stored in the xref holding database. The MySQL database is created from the SQL held in

`ensembl/misc-scripts/xref_mapping/sql/table.sql`

and the metadata required to start using the xref system is held in

`ensembl/misc-scripts/xref_mapping/sql/populate_metadata.sql`

The xref holding database consists of the following tables:

<i>Table name</i>	<i>Purpose</i>
xref	The central table for storing xrefs. Contains attributes which are common to all types of xrefs.
primary_xref	Contains extra columns required for storing xrefs with sequence data. Links to xref table to store common xref data.
dependent_xref	Links pairs of xrefs in a parent-child relationship. Note that the linkage is done by xref_id so the all the actual data is stored in the xref table.
direct_xref	Allows storage of xrefs where the link to Ensembl has already been calculated, e.g. by the maintainers of the xref's source. Links an entry in the xref table to an Ensembl stable ID.
synonym	Links pairs of xrefs (in the xref table) that are synonyms of each other.
source	High-level data about sources from which xref data is obtained.
source_url	Stores the actual URLs from which xref data is downloaded. There are generally several rows in source_url for each source.
species	Species information, including any commonly-used aliases for the species.
interpro	Stores the relationships between Interpro accessions and protein family IDs.

## Parsing xrefs

This is the first of the two stages of creating xrefs. There are separate parsers for each source; while the details of the parsing are different for each one, the overall functionality can be summarised as follows:

- Connect to source FTP or web site
- Download appropriate file(s)
- Compare checksum of file with previous checksum
- If different:
  - Parse the file and load into holding database.
  - Generate xrefs, primary xrefs, dependent xrefs and synonyms as appropriate.

All species have the same "core" set of xrefs which consists of the following:

<i>Source name</i>	<i>Website</i>	<i>Parser(s) used to populate</i>
Uniprot/SPtrembl	<a href="ftp://ftp.ebi.ac.uk/pub/databases/SPproteomes/swissprot_files">ftp://ftp.ebi.ac.uk/pub/databases/SPproteomes/swissprot_files</a>	UniProtParser.pm
Uniprot/Swissprot	<a href="ftp://ftp.ebi.ac.uk/pub/databases/SPproteomes/swissprot_files">ftp://ftp.ebi.ac.uk/pub/databases/SPproteomes/swissprot_files</a>	UniProtParser.pm
Refseq_peptide	<a href="ftp://ftp.ncbi.nih.gov/refseq/">ftp://ftp.ncbi.nih.gov/refseq/</a>	RefSeqGPFFParser.pm
Refseq_dna	<a href="ftp://ftp.ncbi.nih.gov/refseq/">ftp://ftp.ncbi.nih.gov/refseq/</a>	RefSeqParser.pm
UniGene	<a href="ftp://ftp.ncbi.nih.gov/repository/UniGene/">ftp://ftp.ncbi.nih.gov/repository/UniGene/</a>	UniGeneParser.pm
IPI	<a href="ftp://ftp.ebi.ac.uk/pub/databases/IPI/current/">ftp://ftp.ebi.ac.uk/pub/databases/IPI/current/</a>	IPIParser.pm
MIM	secondary <sup>3</sup>	UniProtParser.pm RefSeqGPFFParser.pm
PDB	secondary <sup>3</sup>	UniProtParser.pm
EMBL	secondary <sup>3</sup>	UniProtParser.pm
Protein_id	secondary <sup>3</sup>	UniProtParser.pm
EntrezGene	secondary <sup>3</sup>	RefSeqGPFFParser.pm
GO	<a href="ftp://ftp.ebi.ac.uk/pub/databases/GO/goa/">ftp://ftp.ebi.ac.uk/pub/databases/GO/goa/</a>	GOParser.pm
Interpro	<a href="ftp://ftp.ebi.ac.uk/pub/databases/interpro/interpro.xml.gz">ftp://ftp.ebi.ac.uk/pub/databases/interpro/interpro.xml.gz</a>	InterproParser.pm <sup>1</sup>
PubMed	secondary <sup>3</sup>	UniProtParser.pm <sup>2</sup>

<sup>1</sup> no link to primary\_xref, special case

<sup>2</sup> new data source (not stored in old xref system)

<sup>3</sup> secondary data source see parser to get more information

There are also certain species-specific xref sources:

<i>Species</i>	<i>Source name</i>	<i>Website</i>	<i>Parser</i>
Human	HUGO	<a href="http://www.gene.ucl.ac.uk/publicfiles/nomen">http://www.gene.ucl.ac.uk/publicfiles/nomen</a> ens4.txt + ens1.dat	HUGOParser.pm
Human	OMIM	ftp://ftp.ncbi.nih.gov/repository/OMIM/morbidmap	MIMParser.pm <sup>2</sup>
Human	CCDS	Local file	CCDSParser.pm
Mouse	MarkerSymbol (MGD/MGI)	ftp://ftp.informatics.jax.org/pub/reports/MRK_SwissProt_TrEMBL.rpt	MGDParser.pm
Rat	RGD	ftp://rgd.mcw.edu/pub/data_release/genbank_to_gene_ids.txt	RGDParser.pm
Zebrafish	ZFIN	<a href="http://zfin.org/data_transfer/Downloads/refseq.txt">http://zfin.org/data_transfer/Downloads/refseq.txt</a> <a href="http://zfin.org/data_transfer/Downloads/swissprot.txt">http://zfin.org/data_transfer/Downloads/swissprot.txt</a>	ZFINParser.pm
C Elegans	WormBase	ftp://ftp.sanger.ac.uk/pub/databases/wormpep/wormpep140/wormpep.table140	WormPepParser.pm

The following section gives more details on each of the parsers.

### **UniProtParser.pm**

Each record is stored as an xref and a primary\_xref.

The accession is the main key and is taken from the AC line.

Any DR line that matches a valid source name (in the database) will be processed and stored as xrefs and dependent xrefs. Currently these are MIM, PDB, EMBL.

In addition protein\_id is taken from the EMBL line.

NOTE: InterPro is not loaded here as Interpro does not match InterPro (note capital P here). This is loaded separately via the interpro parser.

New entries are now added for the medline and pubmed lines by parsing RX lines with PUBMED xrefs.

Species specific files are parsed here so all records are stored.

### **RefSeqParser.pm**

Each record is stored as an xref and a primary\_xref.

Only the sequence data and accessions are stored.

### **RefSeqGPFFParser.pm**

Each record is stored as an xref and a primary\_xref.

The accession is the main key and is taken from the AC line.

EntrezGene, OMIM and pubmed are stored as xrefs and dependent xrefs.

Species specific files are parsed here so all records are stored.

### **IPIParser.pm**

Each record is stored as an xref and a primary\_xref.

Only the sequence data and the id are stored.

### **UniGeneParser.pm**

Each record is stored as an xref and a primary\_xref.

The cluster id is the main key and is taken from the /ug field.

### **CCDSParser.pm**

Each record is stored as a direct xref between the ensembl stable id and the CCDS reference.

### **GOParser.pm**

Will only add entries if uniprot or refseq entry has been loaded already.

ENSEMBL entries are also ignored as these will only map onto themselves.

Most GO entries will already exist in the xref table (from Uniprot parsing) but most will not have the description, so this is added if there is none.

Dependent xrefs are created if they do not exist already.

### **InterproParser.pm**

The xrefs are stored for each Interpro but NO dependent xrefs are stored.

Instead a separate table is populated (interpro) with the interpro/pfam mappings. Uniprot/Refseq accessions are NOT checked to see if they are already in the database, therefore is species non-specific, but the xref is stored with the species specified in the run.

### **MIMParser.pm**

Uses the Gene names to map the MIM numbers to the protein accessions via the HUGO numbers. So HUGO has to be parsed already as well as Uniprot and Refseq.

These are stored as OMIM at present and are expected to replace the disease database.

### **HUGOParser.pm, MGDParser.pm, RGDPParser.pm, ZFINParser.pm**

Uniprot and Refseq must be already be parsed. Entries are added to the xref table and the dependent xref linked to the proteins (if they have been loaded). So Entries are added if the accession is valid for uniprot or refseq for that particular species.

### **WormPepParser.pm**

Uniprot and Refseq must be already be parsed. Wormbase file is used to correct the wormbase\_transcript names and add new ones if they do not exist already and a swissprot accession is given.

## IMapping xrefs to Ensembl objects

### Overview

Once the internal xref database has been populated with xref data, the next stage is to map the xrefs to find which Ensembl objects they're associated with. There are three phases to the mapping process:

- Dumping the xref & Ensembl sequences
- Finding the best match(es) between each xref and the Ensembl transcripts/translations etc.
- Creating the relevant entries in the Ensembl core database xref-related tables

### Doing the mapping

The Perl script `xref_mapper.pl` controls the mapping process. More details on actually running this phase of the process are in the tutorial at the end of this document.

### Package structure

The Perl modules that perform the mapping are part of the `XrefMapper` package. The `BasicMapper.pm` module contains most of the functionality for doing the mapping. It is possible to subclass this class in order to allow species-specific modifications to some functionality; in most cases the only method that will need to be overridden is `get_set_lists` which returns a list of lists specifying the xref source(s) and species to map, and the mapping method to be used (see below).

`xref_mapper.pl` looks for species-specific Perl modules in the `XrefMapper` package. Thus if you want to introduce some species-specific functionality, you should put your species-specific `.pm` file (named appropriately, e.g. `homo_sapiens.pm`) in the `XrefMapper` directory, and have it extend `BasicMapper.pm`.

### Mapping methods

There are many different ways to perform a sequence mapping. For this reason it is possible to define your own mapping method (assuming you're going to use `exonerate` to do the mapping) simply by creating a Perl module which extends `XrefMapper::Methods::ExonerateBasic` and overriding the `options()` method to specify which options you want to pass to `exonerate`. There are several pre-defined mapping methods in the `XrefMapper/Methods` directory already.

Note that the `exonerate --bestn 1` option will usually give the best-in-genome result, but it will produce more than one “best” mapping if two equally-good mappings exist, since there is no other way to differentiate them.

### Use of LSF

The mapping itself is done on the farm using LSF. Separate job arrays are submitted for DNA and protein alignments. The number of jobs in each array is calculated before submission so that the whole process takes about 20 minutes to run, in order to make best use of farm resources.

When the mapping is complete the `.map` files produced by all the `exonerate` jobs are parsed and files suitable for loading into the xref-related tables of the core database are produced. If there were errors during the mapping, `.err` files of non-zero length will be produced. The system will check for these and warn if any are found so that you can investigate further.

### Dependent xrefs

The sequence mapping is done between xrefs with DNA/protein sequence and Ensembl transcripts/translations respectively. Of course there are many xrefs which do not have associated sequence. If these have been stored correctly in the internal xref database, any dependent xrefs that are associated with a primary xref will be automatically mapped to the same Ensembl object as the primary xref.

## Loading xref data into the core database

The end result of the mapping process is a set of files containing the xref data and mappings to Ensembl objects in a format suitable for loading into an Ensembl core database. There are two groups of files:

- `.txt` files containing tab-delimited values which are suitable for loading into database tables using the `mysqlimport` utility or the `LOAD DATA INFILE` command from a MySQL client. The part of the filename before the `.txt` is the name of the table into which it should be loaded.
- `.sql` files used to update existing data, e.g. `display_xrefs`

<i>Filename</i>	<i>Purpose</i>
<code>xref.txt</code>	Contains all primary and dependent xrefs that were mapped.
<code>object_xref.txt</code>	Relationships between all xrefs in <code>xref.txt</code> and their associated ensembl objects.
<code>identity_xref.txt</code>	Data about the quality of each object-xref mapping.
<code>external_db.txt</code>	The sources from which the xrefs were taken.
<code>external_synonym.txt</code>	Synonym information.
<code>transcript_display_xref.sql</code>	SQL that can be executed to set the <code>display_xref</code> column of existing transcripts.

Note that in the case of the `.txt` files, internal databases IDs are numbered to start after the highest existing internal ID; this allows you to load xrefs, `object_xrefs` etc into existing tables without the risk of overwriting existing internal IDs.

Note that if you specify the `-upload` option to `xref_mapper.pl`, the data in the files above will automatically be loaded into the core database.

## Tutorial

Two scripts need to be run: `xref_parser.pl` to download and parse the xrefs into the xref holding database, and `xref_mapper.pl` to map them to Ensembl objects.

## Parsing

The Perl script to create and populate the database is `xref_parser.pl`

`xref_parser --help` produces

```
xref_parser.pl -user {user} -pass {password} -host {host} -port {port}
               -dbname {database} -species {species1,species2}
               -source {source1,source2} -skipdownload -create
```

If no source is specified then then all sources are loaded. The same is done for all species so it is best to specify this one or the script may take a while.

Before running the script, look at `populate_metadata.sql`; in particular, towards the end, there is a section which populates the `source_url` table with the locations of the files for each source. Please verify that these are correct and complete for the species in question, *particularly if it is the first time the new xref system has been run for a new species*.

So to load/parse all the xrefs for the human the command would be:-

```
xref_parser.pm -host host1 -port 3350 -user admin -pass password -dbname xref_store
-species human -create
```

Note: Due to the fact that some Uniprot/Refseq entries may no longer be valid (loaded) some xrefs are ignored, so do not worry if you see xref ignored messages with values.

So we now have a set of xrefs that are dependent on the Uniprot and Refseq entries loaded. These can then be mapped to the Ensembl entities with the `xref_mapper.pl` script.

### Note: adding new xref sources

To add a brand new xref source you will have to edit `sql/populate_metadata.sql`.

Add an entry to the source table first, i.e.

```
INSERT INTO source VALUES (2000, 'NEW', 1, 'Y', 4);
```

Because some sources are dependent on others being loaded the last argument is the order. Lower numbers are processed first. i.e. For HUGO to be loaded the Refseq and Uniprot data must already be loaded as these give the list of valid values to load.

You will also have to specify the files to download and the parser to use. i.e.

```
INSERT INTO source_url (source_id, species_id, url, checksum, file_modified_date,
upload_date, parser) VALUES (2000, 9606, 'ftp://ftp.new.org/new.gz', '', now(), now(),
"NEWParser");
```

You will have to create `XrefParser/NEWparser.pm`

## Mapping

The script to create and populate the database is `xref_mapper.pl`

```
xref_mapper --help produces

usage: perl xref_mapper <options>

options:

-file <input_file>    input file with keyword pairs for 'species','host',
                      'port', 'dbname', 'user', 'password' and 'directory'

-maxdump <int>        dump only <int> sequences.

-dumpcheck            only dump if files do not exist.

-location            only dump a subset of the genome. Format:
                      coord_system:version:name:start:end:strand
                      e.g.
                      chromosome:NCBI34:X:1000000:2000000:1
                      start, end, strand are optional
                      coord_system can also be 'seqlevel' or 'toplevel'
                      USE WITH CAUTION - MAY GIVE CONFLICTING RESULTS!

-useexistingmapping    use existing *.map files
```

```

-upload          upload xref, object_xref, identity_xref data, and set
                  display_xrefs for genes and transcripts. Data is written
                  to *.txt etc regardless of whether this option is used.
                  If external_db in core database is empty, it's populated
                  from ../external_db/external_dbs.txt

-deleteexisting   delete existing data from xref, object_xref,
                  identity_xref and external synonym tables. Also set all
                  existing display_xref_id columns in gene and transcript
                  to null.

-help            display this message

```

It is important to get the contents of the input file (`xref_mapper.input` here) correct. There should be an xref section containing details of the xref holding database that you created in the previous step, for example

```

xref
host=ecs4
port=3350
dbname=glenn_human_xref
user=ensadmin
password=ensembl
dir=/nfs/acari/gpl/work/ensembl/misc-scripts/xref_mapping/xref

```

The key/value pairs should be fairly self-explanatory. Note that lines beginning with # are treated as comments.

After the xref section there should be one section for each species you want to map, for example

```

species=homo_sapiens
host=ecs4
port=3350
dbname=glenn_homo_sapiens_core_29_35b
user=ensadmin
password=ensembl
dir=/nfs/acari/gpl/work/ensembl/misc-scripts/xref_mapping/human

```

again this is fairly straightforward. Note however that the species name *must* be of the form genus\_species, e.g. `homo_sapiens`, you cannot use human or other aliases as you could in the parsing step. This is because species-specific mapping options (see later) are specified in files named e.g. `homo_sapiens.pm`.

Once you've set the appropriate values in the file, the script can be run, for example:

```
perl xref_mapper.pl -file xref_mapper.input -upload -deleteexisting
```

The script will dump the xref & Ensembl sequences to `.fasta` files, run exonerate via LSF on the farm, then parse the exonerate output and upload it to the core database (if the `-upload` option is specified).

Times of the various phases, based on the human xref mapping:

Fasta dumping	Approximately 40 minutes
Exonerate mapping	Approximately 20 minutes
Parsing & uploading	Approximately 15 minutes

Note that if for some reason some part of the mapping script needs to be re-run subsequently, the already-dumped `.fasta` files can be used by specifying the `-dumpcheck` option, and the existing exonerate mappings can be used by specifying the `-useexistingmapping` option.



At the end of the process, all the xref data will have been dumped to tablename.txt files, which can then be loaded into the MySQL database. There are also two .sql files produced, which can be executed to set the transcript and gene display xrefs.

*Note that if you do not specify the -upload option, no data will be uploaded to the core database.*

If you *do* specify `-upload`, the data will all be loaded into the core database. The .txt and .sql files are still written to disk.

### **Species-specific configuration**

You can configure species-specific settings by creating or editing the file called homo\_sapiens.pm (or whatever the species is) in the XrefMapper/ directory. You can override the following:

- `get_set_lists` – link mapping methods to species
- `gene_description_filter_regexps` – regexps matching xref descriptions to ignore when building gene descriptions

### **Mapping methods**

If you need to run exonerate with different options, use different query/target thresholds, or even use some other sequence-matching program, you'll need to create a mapping method in XrefMapper/Methods/ and then modify `get_set_lists()` in BasicMapper.pm and/or your species-specific .pm file.