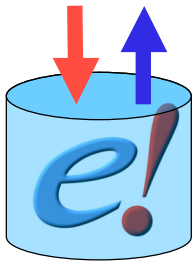
# The Ensembl API



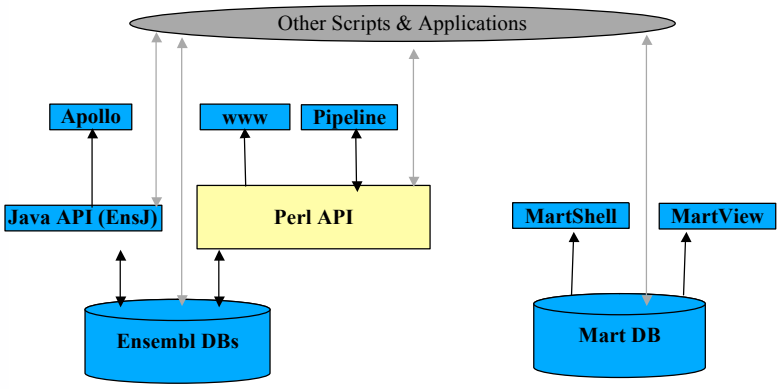
European Bioinformatics Institute (EBI)
Hinxton, Cambridge, UK



## What is the API?

- The Ensembl API (application programming interface) is a framework for applications that need to access or store data in Ensembl's databases.

## System Context






## The Perl API

- Written in Object-Oriented Perl.
- Used to retrieve data from and to store data in Ensembl databases.
- Foundation for the Ensembl Pipeline and Ensembl Web interface.

## Why use an API?

- Uniform method of access to the data.
- Avoid writing the same thing twice: reusable in different systems.
- Reliable: lots of hard work, testing and optimisation already done.
- Insulates developers to underlying changes at a lower level (i.e. the database).

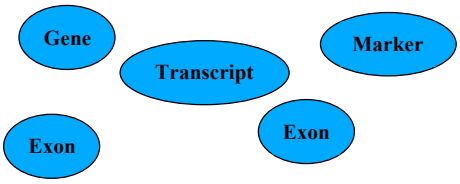


## Main object types

- Data objects
- Object adaptors
- Database adaptors (Registry)

# Data Objects

- Information is obtained from the API in the form of *Data Objects*.
- A *Data Object* represents a piece of data that is (or can be) stored in the database.






# Data Objects – Code Example

```
# print out the start, end and strand of a transcript
print $transcript->start, '-', $transcript->end,
      '(',$transcript->strand, ")\n";

# print out the stable identifier for an exon
print $exon->stable_id, "\n";

# print out the name of a marker and its primer sequences
print $marker->display_marker_synonym->name, "\n";
print "left primer: ", $marker->left_primer, "\n";
print "right primer: ", $marker->right_primer, "\n";

# set the start and end of a simple feature
$simple_feature->start(10);
$simple_feature->end(100);
```

## Object Adaptors

- *Object Adaptors* are factories for *Data Objects*.
- *Data Objects* are retrieved from and stored in the database using *Object Adaptors*.
- Each *Object Adaptor* is responsible for creating objects of only one particular type.
- *Data Adaptor* *fetch*, *store*, and *remove* methods are used to retrieve, save, and delete information in the database.



## Object Adaptors – Code Example

```
# fetch a gene by its internal identifier
$gene = $gene_adaptor->fetch_by_dbID(1234);

# fetch a gene by its stable identifier
$gene =
  $gene_adaptor->fetch_by_stable_id('ENSG0000005038');

# store a transcript in the database
$transcript_adaptor->store($transcript);

# remove an exon from the database
$exon_adaptor->remove($exon);

# get all transcripts having a specific interpro domain
@transcripts =
  @{$transcript_adaptor->fetch_all_by_domain('IPR000980')};
```

## The Registry

- Is a central store of all the adaptors
- Automatically adds Adaptors on creation
- Adaptors can be retrieved by using the species, database type and Adaptor type.
- Enables you to specify configuration scripts
- Very useful for multi species scripts.



## load_registry_from_db

- Loads all the databases for the correct version of this API
  - Ensures you use the correct API and Database
  - Lazy loads so no database connections are made until requested
- Use -verbose option to print out the databases loaded

# Registry Usage

```
use Bio::EnsEMBL::Registry;
my $reg = "Bio::EnsEMBL::Registry";

$reg->load_registry_from_db(
    -host => 'ensembldb.ensembl.org',
    -user => 'anonymous');

$slice_adaptor =
      $reg->get_adaptor("human", "core", "Slice");
```



# Getting More Information

- Inline POD
- Perldoc – viewer for inline API documentation:
  http://www.ensembl.org/info/software/core/index.html
- Tutorial document:
  http://www.ensembl.org/info/software/core/core_tutorial.html
- ensembl-dev mailing list: <ensembl-dev@ebi.ac.uk>

## Question 1

a) Use the function load_registry_from_db to load all the databases and print the names of the databases found?
   - hint: use Perldoc to find the methods to do this

b) What is the database name fro the human core database?
   - hint: use the Registry's get_adaptor method

c) What other core databases are there?



## Perl Debugger

```
$ perl -d 1.pl    # start the perl script in debug mode

    .   b 12           # breakpoint/stop at line 12
    .   l              # list script
    .   x $object      # examine the data in $object
    .   x 2 $object    # examine object down to 2 levels
    .   m $object      # what methods are available
    .   r              # run the script
    .   s              # step to next line following
                       # methods
    .   n              # step to next line in this script
```

# Question 2

a) Fetch the sequence of the first 10MB of chromosome 20 and write the sequence of the chromosome Slice to file in FASTA format.
   - hint: Slice objects inherit from Bio::Seq so can be written to file easily using Bio::SeqIO, ie:

```
my $output = Bio::SeqIO->new( -file=>'>filename.fasta',
    -format=>'Fasta');
$output->write_seq($slice);
```

b) Use the identifier 'ENSG00000101266' to fetch a slice surrounding this gene with 2kb of flanking sequence.

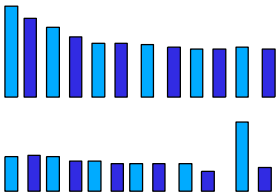
c) Print out information about your slices, include the name of the coordinate system, the name of the sequence, the start and end coordinates of the slice and strand.

d) Determine the number of genes on the first 10MB of chromosome 20.

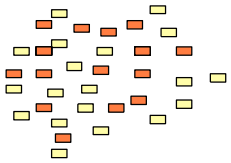


# Coordinate Systems

- Ensembl stores features and sequence in a number of coordinate systems.
- Example coordinate systems: *chromosome*, *clone*, *scaffold*, *contig*



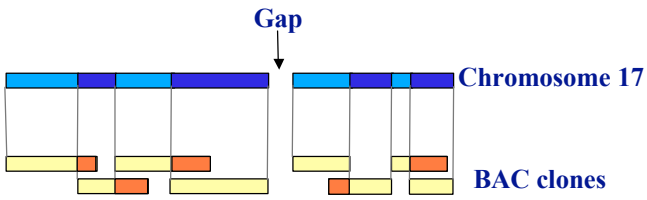
**Chromosome CoordSystem**
**1, 2, ..., X, Y**



**Clone CoordSystem**
**AC105024, AC107993, etc.**

# Coordinate Systems

- Regions in one coordinate system may be constructed from a tiling path of regions from another coordinate system.






# Coordinate Systems – Code Example

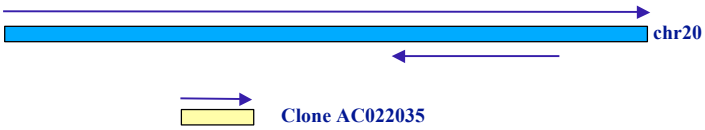
```
# get a coordinate system adaptor
$csa = Bio::EnsEMBL::Registry->get_adaptor('human',
    'core', 'coordsystem');

# print out all of the coord systems in the db
$coord_systems = $csa->fetch_all;
foreach $cs (@$coord_systems) {
  print $cs->name, ' ',$cs->version, "\n";
}
```

Sample output:

```
chromosome NCBI36
supercontig
clone
contig
```

# Slices

- A *Slice* Data Object represents an arbitrary region of a genome.
- *Slices* are not directly stored in the database.
- A *Slice* is used to request sequence or features from a specific region in a specific coordinate system.







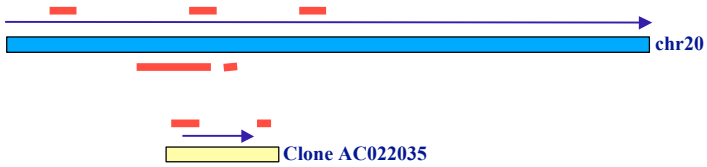


# Slices – Code Example

```
# get the slice adaptor
$slice_adaptor = $reg->get_adaptor('human', 'core', 'slice');

# fetch a slice on a region of chromosome 12
$slice = $slice_adaptor->fetch_by_region('chromosome', '12',
                                         1e6, 2e6);

# print out the sequence from this region
print $slice->seq, "\n";

# get all clones in the database and print out their names
@slices = @{ $slice_adaptor->fetch_all('clone') };
foreach $slice (@slices) {
  print $slice->seq_region_name, "\n";
}
```

# Features

- *Features* are Data Objects with associated genomic locations.
- All *Features* have *start*, *end*, *strand* and *slice* attributes.
- *Features* are retrieved from Object Adaptors using limiting criteria such as identifiers or regions (slices).






# Features in the database

- Gene
- Transcript
- Exon
- PredictionTranscript
- PredictionExon
- DnaAlignFeature
- ProteinAlignFeature
- SimpleFeature
- MarkerFeature
- QtlFeature
- MiscFeature
- KaryotypeBand
- RepeatFeature
- AssemblyExceptionFeature
- DensityFeature

# A Complete Code Example

```
use Bio::EnsEMBL::Registry;
my $reg = "Bio::EnsEMBL::Registry";

$reg->load_registry_from_db(
    -host => 'ensembldb.ensembl.org',
    -user => 'anonymous');

$slice_ad = $db->get_adaptor('human', 'core', 'slice');
$slice = $slice_ad->fetch_by_region('chromosome', 'X',
                                    1e6, 10e6);

foreach $sf (@{ $slice->get_all_SimpleFeatures }) {
  $start  = $sf->start;
  $end    = $sf->end;
  $strand = $sf->strand;
  $score  = $sf->score;
  print "$start-$end ($strand) $score\n";
}
```



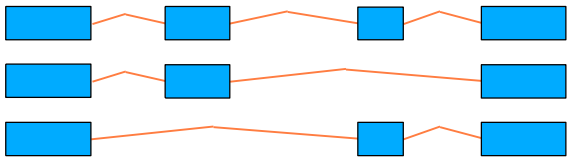
# Question 3

a) Get all the repeat features from chromosome 20:1-500k. Print out the name and position of each and the total number.

b) Get a protein alignment feature with the db identifier 34425. What is the analysis program that produced this alignment and what are the two sequences called.
   - ✓ Hint: use a ProteinAlignFeatureAdaptor
   - ✓ print the results of fetch_by_dbID

# Genes, Transcripts, Exons

- Genes, Transcripts and Exons are objects that can be used just like any other Feature object.
- A Gene is a set of alternatively spliced Transcripts.
- A Transcript is a set of Exons.





## Genes, Transcripts, Exons – Code Example

```
# fetch a gene by its stable identifier
$gene = $gene_adaptor->fetch_by_stable_id('ENSG00000123427');

# print out the gene, its transcripts, and its exons
print "Gene: ", get_string($gene), "\n";
foreach $transcript (@{ $gene->get_all_Transcripts }) {
  print "  Transcript: ", get_string($transcript), "\n";
  foreach $exon (@{ $transcript->get_all_Exons }) {
    print "    Exon: ", get_string($exon), "\n";
  }
}

# helper function: returns location and stable_id string
sub get_string {
  $feature = shift;
  $stable_id  = $feature->stable_id;
  $seq_region = $feature->slice->seq_region_name;
  $start = $feature->start;
  $end = $feature->end;
  $strand = $feature->strand;
  return "$stable_id  $seq_region:$start-$end($strand)";
}
```

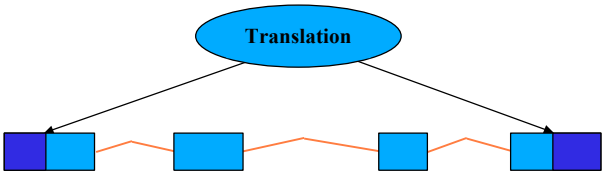
# Genes, Transcripts, Exons – Sample Output

```
Gene: ENSG00000123427  12:56452650-56462590(1)
  Transcript: ENST00000300209  12:56452650-56462590(1)
    Exon: ENSE00001180238  12:56452650-56452951(1)
    Exon: ENSE00001354204  12:56453067-56453178(1)
    Exon: ENSE00001354199  12:56460305-56462590(1)
  Transcript: ENST00000337139  12:56454741-56462451(1)
    Exon: ENSE00000838990  12:56454741-56454812(1)
    Exon: ENSE00001246483  12:56454943-56454949(1)
    Exon: ENSE00001141252  12:56460305-56462451(1)
  Transcript: ENST00000333012  12:56452728-56462590(1)
    Exon: ENSE00000838993  12:56452728-56452951(1)
    Exon: ENSE00001354204  12:56453067-56453178(1)
    Exon: ENSE00001246506  12:56454679-56454817(1)
    Exon: ENSE00001336722  12:56460305-56462590(1)
```



# Translations

- Translations are not Features.
- A Translation object defines the UTR and CDS of a Transcript.
- Peptides are not stored in the database, they are computed on the fly using Translation objects.



# Translations – Code Example

```
# fetch a transcript from the database
$transcript =
  $transcript_adaptor->fetch_by_stable_id('ENST00000333012');

# obtain the translation of the transcript
$translation = $transcript->translation;

# print out the translation info
print "Translation: ", $translation->stable_id, "\n";
print "Start Exon: ",$translation->start_Exon->stable_id,"\n";
print "End Exon:   ", $translation->end_Exon->stable_id, "\n";

# start and end are coordinates within the start/end exons
print "Start : ", $translation->start, "\n";
print "End   : ", $translation->end, "\n";

# print the peptide which is the product of the translation
print "Peptide : ", $transcript->translate->seq, "\n";
```



# Translations – Sample Output

```
Translation: ENSP00000327425
Start Exon: ENSE00000838993
End Exon:   ENSE00001336722
Start : 3
End   : 22
Peptide :
SERRPRCPGTPLAGRMADPGPDPESESESVFPREVGLFADSYSEKSQFCFCGHVLTITQN
FGSRLGVAARVWDAALSLCNYFESQNVDFRGKKVIELGAGTGIVGILAALQGAYGLVRET
EDDVIEQELWRGMRGACGHALSMSTMTPWESIKGSSVRGGCYHH
```

## Question 4

a) Fetch a gene with id 'ENSG00000101266', print information about the number of transcripts and exons.

b) For the above gene, get all the transcripts and list the number of exons in each and the translations.

c) Why do the exon numbers not match?

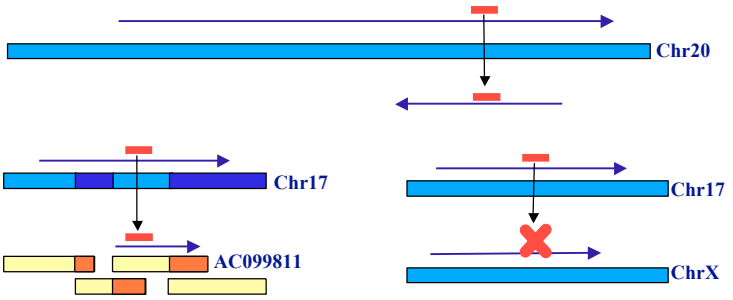


## Coordinate Transformations

- The API provides the means to convert between any related coordinate systems in the database.
- Feature methods *transfer*, *transform*, *project* can be used to move features between coordinate systems.
- Slice method *project* can be used to move features between coordinate systems.

# Feature::transfer

- The Feature method *transfer* moves a feature from one Slice to another.
- The Slice may be in the same coordinate system or a different coordinate system.






# Feature::transfer – Code Example

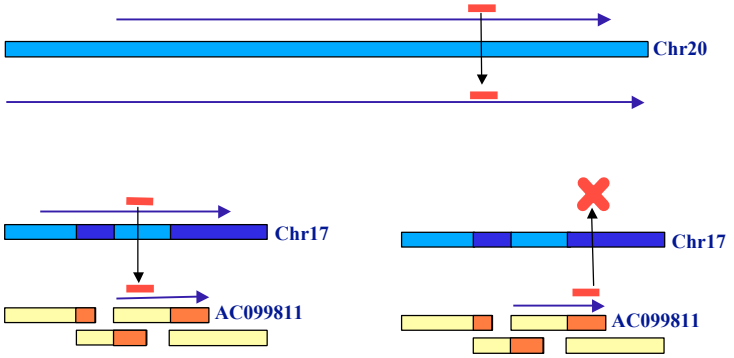
```
# fetch an exon from the database
$exon = $exon_adaptor->fetch_by_stable_id('ENSE00001180238');
print "Exon is on slice: ", $exon->slice->name, "\n";
print "Exon coords: ", $exon->start, '-', $exon->end, "\n";

# transfer the exon to a small slice just covering it
$exon_slice = $slice_adaptor->fetch_by_Feature($exon);
$exon = $exon->transfer($exon_slice);

print "Exon is on slice: ", $exon->slice->name, "\n";
print "Exon coords: ", $exon->start, '-', $exon->end, "\n";
```

Sample output:

```
Exon is on slice: chromosome:NCBI35:12:1:132449811:1
Exon coords: 56452650-56452951
Exon is on slice:
chromosome:NCBI35:12:56452650:56452951:1
Exon coords: 1-302
```

# Feature::transform

- *Transform* is like the *transfer* method but it does not require a Slice argument, only the name of a Coordinate System. Slice will span entire CoordSystem.






# Feature::transform – Code Example

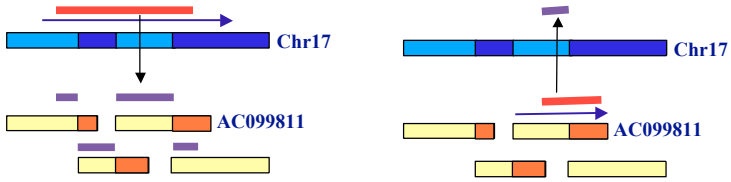
```
# fetch a prediction transcript from the database
$pta = $reg->get_adaptor('human', 'core',
    'PredictionTranscript');
$pt  = $pta->fetch_by_dbID(1834);
print "Prediction Transcript: ", $pt->stable_id, "\n";
print "On slice: ", $pt->slice->name, "\n";
print "Coords: ", $pt->start, '-', $pt->end, "\n";

# transform the prediction transcript to the supercontig system
$pt = $pt->transform('supercontig');

print "On slice: ", $pt->slice->name, "\n";
print "Coords: ", $pt->start, '-', $pt->end, "\n";
```

Sample output:

```
Prediction Transcript: GENSCAN00000001834
On slice: contig::AL390876.19.1.185571:1:185571:1
Coords: 131732-174068
On slice: supercontig::NT_008470:1:40394264:1
Coords: 10814989-10857325
```

# Feature::project

- *Project* does not move a feature, but it provides a definition of where a feature lies in another coordinate system.
- It is useful to determine where a feature that spans multiple sequence regions lies.






# Feature::project – Code Example

```
# fetch a transcript from the database
$tr =
  $transcript_adaptor->fetch_by_stable_id('ENST00000351033');

$cs = $tr->slice->coord_system;
print "Coords: ", $cs->name, ' ',
      $tr->slice->seq_region_name,' ', $tr->start, '-',
      $tr->end, '(', $tr->strand, ")\n";

# project to the clone coordinate system
foreach $segment (@{ $tr->project('clone') }) {
  ($from_start, $from_end, $pslice) = @$segment;

   print "$from_start-$from_end projects to clone region: ";
   print $pslice->seq_region_name, ' ', $pslice->start,
        '-', $pslice->end, "(", $pslice->strand, ")\n";
}
```

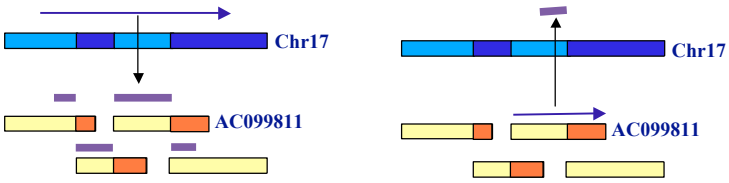
# Feature::project – Sample Output

```
Coords: chromosome 1 147647-147749(-1)
1-103 projects to clone region: AL627309.15 147271-147373(-1)
```



# Slice::project

- *Slice::project* acts exactly like Feature::project, except that it works with a Slice instead of a Feature.
- It is used to determine how one coordinate system is made of another.



## Slice::project – Code Example

```
# fetch a slice on a region of chromosome 5
$slice = $slice_adaptor->fetch_by_region('chromosome', '5',
                                         10e6, 12e6);

# project to the clone coordinate system
foreach $segment (@{ $slice->project('clone') }) {
  ($from_start, $from_end, $pslice) = @$segment;

   print "$from_start-$from_end projects to clone region: ";
   print $pslice->seq_region_name, ' ', $pslice->start,
        '-', $pslice->end, "(", $pslice->strand, ")\n";
}
```



## Slice::project – Sample Output

```
1-55893 projects to clone region: AC010629.8 24890-80782(1)
55894-151517 projects to clone region: AC091905.3 423-96046(1)
151518-296668 projects to clone region: AC034229.4 768-145918(1)
296669-432616 projects to clone region: AC012640.12 1-135948(-1)
432617-562215 projects to clone region: AC092336.2 1-129599(-1)
562216-646577 projects to clone region: AC112200.3 1-84362(-1)
646578-722633 projects to clone region: AC106760.2 2135-78190(1)
722634-889200 projects to clone region: AC012629.9 1-166567(-1)
889201-1030964 projects to clone region: AC010626.6 18549-160312(1)
1030965-1148679 projects to clone region: AC005610.1 105694-
223408(1)
1148680-1286532 projects to clone region: AC004648.1 1-137853(-1)
1286533-1367657 projects to clone region: AC005367.1 463-81587(1)
1367658-1488052 projects to clone region: AC003954.1 1-120395(-1)
1488053-1522377 projects to clone region: AC004633.1 1901-36225(1)
1522378-1776593 projects to clone region: AC010433.9 4714-258929(1)
1776594-1950687 projects to clone region: AC113390.3 22452-196545(1)
1950688-1994121 projects to clone region: AC127460.2 1-43434(-1)
1994122-2000001 projects to clone region: AC114289.2 164862-170741(-
1)
```

# Question 5

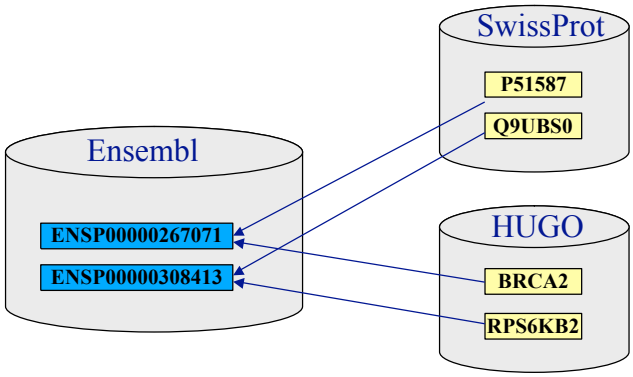
a) Fetch any gene from clone 'AL049761.11', print details of the gene including the coordinate system and start/end positions.
   - Hint: to get a gene, first fetch a slice containing the clone with fetch_by_region, then get a gene.

b) Transform the gene to 'toplevel' and print the new details.



# External References

- Ensembl cross references its Gene predictions with identifiers from other databases.



# External References – Code Example

```
# fetch a transcript using an external identifier
my ($tr) =
  @{$transcript_adaptor->fetch_all_by_external_name('TTN')};

print "TTN is Ensembl transcript ", $transc->stable_id, "\n";

# get all external identifiers for a translation
$transl = $tr->translation;
print "External identifiers for ", $transl->stable_id, ":\n";
foreach my $db_entry (@{ $transl->get_all_DBEntries }) {
  print $db_entry->dbname, ":", $db_entry->display_id, "\n";
}
```



# External References – Sample Output

```
TTN is Ensembl transcript ENST00000286104
External identifiers for ENSP00000286104:
GO:GO:0005524
GO:GO:0016020
GO:GO:0005975
GO:GO:0006979
GO:GO:0004674
GO:GO:0006468
GO:GO:0004713
GO:GO:0004896
GO:GO:0007517
GO:GO:0008307
Uniprot/SPTREMBL:Q10465
EMBL:X90569
protein_id:CAA62189.1
GO:GO:0006941
GO:GO:0030017
GO:GO:0004601
Uniprot/SPTREMBL:Q8WZ42
EMBL:AJ277892
protein_id:CAD12456.1
```

## Question 6

a) Fetch a database entry for the Refseq_dna gene NP_808227.

b) Fetch the equivalent ENSEMBL gene
   - Hint: use a gene adaptor and fetch_all_by_external_name()

c) Print out all the other linked database entries for the gene.



## Getting More Information

- Inline POD
- Perldoc – viewer for inline API documentation:
  http://www.ensembl.org/info/software/core/index.html
- Tutorial document:
  http://www.ensembl.org/info/software/core/core_tutorial.html
- ensembl-dev mailing list: <ensembl-dev@ebi.ac.uk>

# Acknowledgements


- The Ensembl Core Team
  - Glenn Proctor
  - Ian Longden
  - Patrick Meidl
- The rest of the Ensembl team